Coordinating the Web of Services for a Smart Home

Kaldeli, Eirini; Warriach, Ehsan Ullah; Lazovik, Alexander; Aiello, Marco

Published in:
Acm transactions on the web

DOI:
10.1145/2460383.2460389

Publication date:
2013

Coordinating the Web of Services for a Smart Home

EIRINI KALDELI, EHSAN ULLAH WARRIACH, ALEXANDER LAZOVIK, and MARCO AIELLO, University of Groningen

Domotics, concerned with the realization of intelligent home environments, is a novel field which can highly benefit from solutions inspired by service-oriented principles to enhance the convenience and security of modern home residents. In this work, we present an architecture for a smart home, starting from the lower device interconnectivity level up to the higher application layers that undertake the load of complex functionalities and provide a number of services to end-users. We claim that in order for smart homes to exhibit a genuinely intelligent behavior, the ability to compute compositions of individual devices automatically and dynamically is paramount. To this end, we incorporate into the architecture a composition component that employs artificial intelligence domain-independent planning to generate compositions at runtime, in a constantly evolving environment. We have implemented a fully working prototype that realizes such an architecture, and have evaluated it both in terms of performance as well as from the end-user point of view. The results of the evaluation show that the service-oriented architectural design and the support for dynamic compositions is quite efficient from the technical point of view, and that the system succeeds in satisfying the expectations and objectives of the users.

Categories and Subject Descriptors: C.2.4 [**Computer Systems Organization**]: Distributed Systems—*Distributed applications*; H.3.5 [**Information Systems**]: Online Information Services—*Web-based services*

General Terms: Design, Experimentation, Human Factors

Additional Key Words and Phrases: Service-oriented architecture, service composition, internet of things

ACM Reference Format:
Kaldeli, E., Warriach, E. U., Lazovik, A., and Aiello, M. 2013. Coordination the web of services for a smart home. ACM Trans. Web 7, 2, Article 10 (May 2013), 40 pages.
DOI: http://dx.doi.org/10.1145/2460383.2460389

1. INTRODUCTION

The Web, originally intended as a multimedia infrastructure for the Internet, in its constant growth and evolution has called for ever-more sophisticated techniques and architectures to deliver advanced functionality to end-users. Most of these innovative approaches have actually an impact that goes beyond the conventional Web, and influence the technologies used in other open and heterogeneous networks of autonomous entities. A prominent example is the Web of Things in domotic environments, which is concerned with the interoperation of autonomous appliances in order to realize added-value functionalities that enhance the feeling of comfort and safety of the home

This work is supported by the EU project Smart Homes for All (http://www.sm4all-project.eu), contract FP7-22433.
Authors' addresses: E. Kaldeli (corresponding author), E. U. Warriach, A. Lazovik, and M. Aiello, Distributed Systems Group, Johan Bernoulli Institute, University of Groningen, Nijenborgh 9, 9747AG Groningen, The Netherlands; email: kaldeli@gmail.com.

DOI: http://dx.doi.org/10.1145/2460383.2460389

inhabitants. The similarities between the two sorts of Web include the high heterogeneity of the connected nodes, the importance of communication protocols, the support for a dynamic environment where nodes come and leave, and the need to effectively coordinate and integrate the different components, in order to deliver to the end-user a transparent and satisfactory access to the system.

Given these general requirements for the Web of Things, it is natural to look at Service-Oriented Architectures (SOA) as a sound approach to support interoperability, asynchronous communication, late binding of components, and deal with a constantly changing context. One might argue that these issues have been known for long in the area of domotic systems, and there are indeed a number of platforms for pervasive applications that base on the notion of service, such as the Java-based Jini[1] infrastructure or the Universal Plug and Play (UPnP)[2]standard. However, these platforms focus mainly on aspects related to basic device interoperation and spontaneous networking, without providing for dynamic coordination and the realization of more complex and intelligent functionalities that can be built at a higher application level. The aim of the approach presented herein is to take full advantage of the capabilities offered by a well-designed service-oriented architecture for the home, and, by automatically composing the available autonomous device operations, to deliver added-value services which will be perceived as *smart* by the user.

1.1. Smartness via Service Composition

To satisfy the wishes of the user and guarantee his comfort and safety, the house has to be able to exhibit quite complex functionalities rather than just triggering some single service or a predesigned sequence of fixed services. A trivial operation such as turning on a light in a corridor can be achieved with a switch or a passive infrared sensor. However, the coordination of the home so as to effectively deal with a gas leak detection is far more demanding, especially when considering the many possible contextual states the house and the user can be in, each of which may require several possible handlings to achieve the same ultimate safety goal. Moreover, developing rigid solutions that are tailored to a specific home instance and user needs is not an efficient approach, given the considerable effort that is required to adapt them for new customers.

Designing and predicting all possible service compositions is thus not a viable solution given: (1) the large variety of different user requirements and home instances, and (2) the lifecycle of a specific home: devices evolve over time, with new functionalities constantly appearing or disappearing, the state of the devices constantly changes, users move around, and thus the number of possible contextual states can be very high. Therefore, approaches to service coordination in such a dynamic setting have to be easily customizable to different home instances and user needs, be able to support home evolution, and perform complex reasoning about contextual information rather than relying on hardwired sets of service instances. These remarks are in line with the challenges that should be addressed by service composition mechanisms in pervasive environments as identified by Bronsted et al. [2010] in their survey about service composition issues in pervasive computing.

—*Context awareness.* "A composition is context aware if it is sensitive to context changes, its constituent services are sensitive to context changes, or the set of composed services varies with context changes."

[1]www.jini.org.
[2]www.upnp.org.

—*Managing contingencies.* "In a pervasive computing environment, devices—and thus services—often have unpredictable availability [. . .] A service in a given composition might become unavailable and need replacement, so the logic for discovering and inserting a replacement service shouldn't reside in the constituent services."
—*Leveraging heterogeneous devices.* "Pervasive computing systems use a variety of devices, from Internet servers to networked sensors and actuators."
—*Empowering users.* "Pervasive computing applications require new interaction models because document-centric, desktop-based computer interfaces are often unavailable or impractical."

Our approach to service composition aims at addressing these challenges by applying novel AI planning techniques to state-of-the-art pervasive system architectures. This builds on our initial experience on supporting service composition in the logistics and travel domains [Aiello et al. 2002; Lazovik et al. 2003], and has called for a major rethinking and redesign of the adopted methods in order to address domain independence, dealing with numeric values, extended goals, and dynamicity considering an evolving set of elements typical of the Web of Things. In the approach we present here, the objective that a composition has to achieve is described in the form of a declarative goal, while the services are selected and combined during runtime. This way, different compositions can be be computed for the same goal depending on the current state of the devices, thus meeting the first challenge about context awareness. Regarding the second requirement, the system supports dynamic service availability, and devices can enter or leave the network. Since the composition is computed at runtime rather than at design time, the reasoning is performed on the most up-to-date set of services, which may change over time. The third challenge is realized by an open and dynamic pervasive layer which supports a number of different communication protocols, and offers a standard interface for controlling all devices to the upper layers, thus ensuring interoperability. Finally, the infrastructure is user centric and can be easily adapted to match new user needs through the specification of goals which can be inserted either by the designer or the home inhabitants themselves, as well as the support for different user interfaces, such as a touch screen, voice-based, or Brain Computer Interfaces (BCI).

As far as we are aware, this is the first attempt to fully integrate a domain-independent planning component in an event-driven service-oriented prototype for pervasive applications in a domotic environment. In such a setting, the planning module has to continuously interact through asynchronous message passing with the other components, such as the context awareness and the service repository, so that it seamlessly adjusts the planning domain instance to reflect environmental changes, and reacts accordingly at runtime. Although the implementation presented herein relies on the Open Services Gateway initiative (OSGi)[3] for exposing devices as Web services, the architectural components are loosely coupled with each other and independent of the particularities of the specific architecture (e.g., SOAP, OSGi UPnP). Thus, they can be easily adapted so that they inter-work in a different setting (e.g., see Caruso et al. [2012] for an implementation using REST for inter-communication, and custom proxies for a variety of real hardware devices with different protocols). The platform has been fully implemented, evaluated, and tested with real users on a simulated home environment. The technical and user evaluation confirms that the application of AI planning techniques is a viable and realistic approach towards advancing the intelligence of tomorrow's homes, and that the overall system succeeds in satisfying the objectives of users with diverse needs.

[3]www.osgi.org.

The planner used in the domotic architecture is based on CSP (Constraint Satisfaction Problem) solving. The adopted planning domain modeling relies solely on individual descriptions of decoupled device operations, described and implemented using existing protocol standards. This way, the domain representation is kept generic and can serve a variety of different user needs through the configuration of declarative goals, unlike most other approaches which use planning for Web service composition, and require the specification of procedural templates, for example, in HTN [Au et al. 2005] or Golog [Sohrabi et al. 2006]. The domain encoding is based on a variable-based rather than a predicate-based representation, thus maintaining a close relation to the actual way that device operations are realized, for example, adhering to a direct mapping between UPnP- and planning-level variables. These characteristics contribute towards reducing the manual effort and making more intuitive the task of converting the pervasive-level domain and context into their planning-level equivalent, as well as the users' goal specification. Moreover, the CSP-based planner is endowed with a number of nonclassical features, which collectively leverage the range of scenarios that can be effectively dealt with [Kaldeli et al. 2011]. These include the effective handling of numeric-valued fluents, which are common in domotic environments, support for extended goals, dynamic and efficient incorporation of contextual changes in form of constraints, replanning for failure recovery, and dealing with incomplete knowledge and sensing. Moreover, the generated plans are characterized by a high degree of parallelism, which can be exploited at execution time for achieving better overall response times.

1.2. Content and Organization

The present article describes the architecture we have designed, implemented, and evaluated in the context of the European project Smart Homes for All (SM4ALL), with special emphasis on service composition. The architecture is based on SOA principles, where the service orientation is not only at the lower levels of the pervasive layer, but also at the application layer (http://www.sm4all-project.eu). By building on established standards such as OSGi and UPnP, the platform abstracts to higher layers that support complex reasoning on the set of available services and their state, as well as the interaction with state-of-the-art user interfaces such as BCIs. Most importantly, the architecture supports the performance of runtime service composition. A fully working prototype has been implemented and evaluated with respect to the efficiency of the composition layer, but also the acceptability and effectiveness from the side of end-users from diverse backgrounds, namely a group of elderly and disabled people, and a group of younger technologically experienced users. The manuscript extends the results presented at ICSOC'10 [Kaldeli et al. 2010] and contains evaluation of the software presented at the demonstration sessions of ICSOC 2009 and 2010 [Lazovik et al. 2009; Warriach et al. 2010]. The formal methods and AI techniques used in the approach and overviewed herein have been published in AI-related conferences [Kaldeli et al. 2009; 2011].

The remainder of the article is organized as follows. We complete this section by describing some scenarios that seek to demonstrate what kind of problems and situations a "smart" home equipped with the SM4ALL architecture can deal with. In the next section, we introduce the main aspects of the SM4ALL project and the software architecture we propose. Section 3 digs into the details of service composition to achieve smart home behavior. The prototype we built to test the validity of the approach is presented in Section 4, while the engineering process to deliver a smart home is illustrated in Section 5. The technical evaluation of the system and the user evaluation of it are presented in Sections 6 and 7, respectively. Related work is discussed in Section 8 and final remarks are presented in Section 9.

1.3. A Day in the Smart Home

Let us now describe how a smart home equipped with the SM4ALL architecture behaves over a possible course of events that happen throughout a rather adventurous day in the house, including both conventional user requests and reactions to emergency situations. The following scenarios play the role of demonstrative examples throughout the article, and have been tested in a simulated environment as described in Section 6. We consider that the home inhabitant is a disabled person who can move around on an electric wheel-chair, while a nurse pays a visit for some hours every day. A location component keeps track of the location of the users to the level of some predefined areas.

At 8 pm the waking-up goal prescribed by the user is automatically triggered: the alarm clock rings, the curtains in the bedroom are opened, the lights may be turned on depending on the amount of daylight detected by a natural light sensor, and the motorized bed is elevated. After taking a shower, the user wants to move to the sitting room and watch some TV. Such a goal dictates that the TV is set to the user's channel of preference, the lights are adjusted depending on the indication of the natural light sensor, and the curtains are also shut accordingly. The air-conditioner is turned on if the temperature sensor in the living room indicates that the temperature is too high, while the necessary doors are opened to facilitate the user moving to the sitting room.

At noon, the user goes to the kitchen to prepare something to eat. While being there, the smoke detector in the kitchen identifies a potentially dangerous smoke leak—but fortunately not due to fire. As a result, a predefined home goal for dealing with this situation is automatically triggered: after having ensured that the user has safely moved out of the kitchen (let's say to the adjacent sitting room), the door leading to the kitchen is closed to isolate the smoke in a single room. The ventilator is turned on and the kitchen window is also opened, so that the foul air is expelled, while an alarm notification appears on the TV screen. While waiting in the sitting room, the user wants to move back to the kitchen, but only after having been assured that the environment there is safe, and the smoke has been eliminated. This wish implies resorting to sensing to identify the current situation in the kitchen. Let's assume that after some time the smoke is eliminated, causing the alarm on the TV and the ventilator to automatically turn off.

After verifying that no serious damage has been caused, the user moves to the sofa in the sitting room and wishes to have a cold beer in his hands. Assuming that the house is equipped with a housekeeping robot (similar to the cooking assistant described in Gravot et al. [2006]) able of performing basic recognition and manipulation tasks, such as moving around, getting and putting items at particular places, sensing their temperature, etc., then the request of the user can be fulfilled by the robot. Let's say that there are no beers in the fridge, however, the system finds out that there are some beers left on the storage shelf; the assumption is that items in the house are labeled by RFID tags, and a smart fridge and smart shelves keep track of them. Having this information in hand, the robot will move to the storage room and get a beer from there. In order to satisfy the requirement that the beer should be cold, it will proceed in placing the beer it has taken in the fridge, and leave it there for two minutes to cool. Then it will take it out again and bring it to the sofa.

Later in the afternoon, while the user is taking a bath, and the nurse has gone out for some shopping, a fall is identified by the fall detector attached to him [Aiello and Dustdar 2008], and an emergency goal is automatically triggered: the health center is notified and an informative message is sent to the nurse's PDA or mobile phone, while the robot is moved to the bathroom in case the user wants to ask for some additional assistance. Given that the fall has not caused any serious trouble, the night finds the user lying in his bed reading a book, and after some time he decides that it's time for

going to sleep. He thus issues a goal that prepares the bedroom conditions for sleeping, which involves setting the alarm clock to some preferred wake-up time, lowering the motorized bed position, turning off the lights, and closing the curtains.

It should be emphasized that user goals as well as the description of the device functionalities are kept as decoupled as possible from the particular setting of a home instance, and the set of desired service invocations is reasoned at runtime, depending on the capabilities of the particular house and its current context. Thus, the functionality for sending a message to the nurse, for example, is specified in a generic way, so that it may be taken care by different atomic device instances or a combination of them. This depends on which particular devices that can offer the semantically prescribed unified messaging possibility are available in the specific pervasive system, for example, a smart phone, PDA, mobile, etc. Moreover, depending on what is inferred about the current state of the house, the same goal may lead to quite different compositions of activities. For example, regarding the goal about getting a cold beer, if there exists a beer already in the fridge, the composition will instruct the robot to directly get it from there, or, in the case of the fall detection goal, if the nurse happens to be at home, all that has to be done is to turn on a local alarm to notify him, so that he can take care.

Replanning for basic failure recovery. The aforementioned scenarios assume that no contingencies occur during execution, and that all service invocations complete successfully. What if, however, a service is out of order, and responds with a failure or if a timeout occurs? In such cases, the system will first try to reinvoke the erroneous service, and if again a failure or timeout is observed, it will perform replanning. This means that the composition engine will attempt to achieve the goal by computing an alternative plan which does not include the faulty service.

Considering the scenario with the beer described before, let us assume that the door that leads to the kitchen is blocked, for example, because in the meantime someone has put an obstacle which hinders its opening, and therefore the respective automatic door opening invocation reports an error (or times out). As a result, the composition engine looks whether there is an alternative route to the kitchen which does not go through the problematic door. It should be noted that the new plan will take into account the contextual situation that has resulted after executing all actions that preceded the attempt for opening the kitchen door, which means that the robot may need to go back in order to follow the right route. If no alternative plan can be found, then the system responds that the goal is not satisfiable under the given contextual circumstances. To give another example, let us assume that, when executing the plan that prepares the living room for watching TV, the automatic turning on operation of the TV service responds with a failure. Assuming that the robot assistant is also endowed with the capability of turning on the TV by manually pressing the button on the device, the composition engine will compute an alternative plan which involves moving the robot in front of the TV so that it can switch it on.

2. SMART HOMES FOR ALL

The SM4ALL project is a European Union Small and Targeted Research Project, which focuses on the development of an innovative middleware platform for inter-working of smart embedded services based on the concept of service. In this context, it makes use of composability and semantic techniques, in order to guarantee dynamicity, dependability, and scalability, while preserving the privacy and security of the home and its users. The ultimate objective is to realize a domotic infrastructure that is highly interactive and adaptive to different houses and users. The key idea underpinning the SM4ALL architecture is that the software infrastructure is entirely based on the abstraction of a service, providing for an open, dynamic, and flexible sensing and control

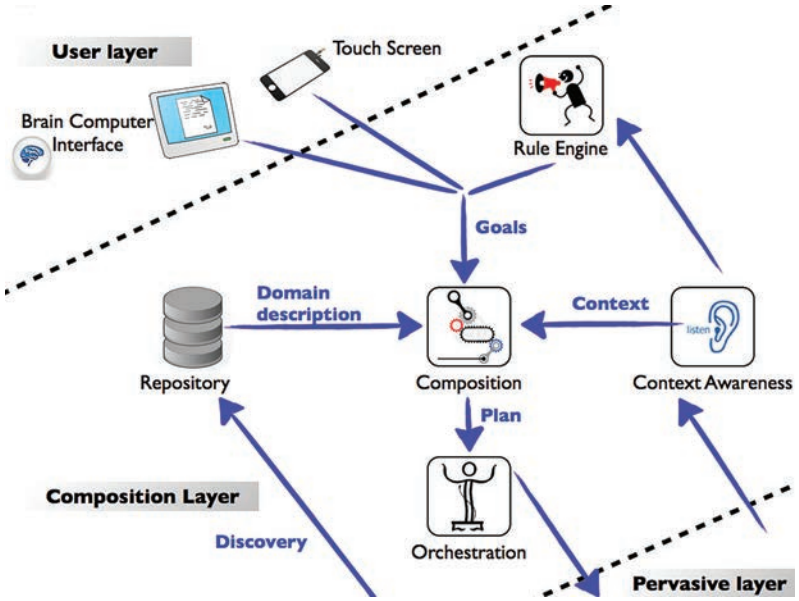

Fig. 1. Architectural overview.

infrastructure. Figure 1 provides a schematization of the systems' main components and their basic interactions. One can distinguish three macrolayers: the pervasive layer, where the home devices live; the composition layer, which is responsible for collecting information about the environment, interpreting it, and coordinating the available services; and the user layer which provides the interface for issuing commands to the home. The technologies and methodologies used for realizing each layer may vary, for example, in the final real apartment setting at Fondazione Santa Lucia in Rome, the Representational State Transfer (REST) protocol is used, while the composition layer also includes an offline synthesis engine with prespecified target services [Berardi et al. 2005]. In the following, we describe an implementation of the architecture that bases on simulating the devices in OSGI UPnP and applying a CSP-based AI planner for achieving dynamic compositions by inter-communicating with the other components.

The Pervasive Layer

The role of the pervasive layer is to discover and interconnect networked devices, and provide a common mechanism for accessing the services they offer for the rest of the middleware layers and applications. All types of devices are described in a standardized programmatic manner, and are controlled in accordance with this description. The pervasive platform enables heterogeneous components to be integrated independently of their interconnectivity protocol, through the use of an appropriate proxy for each communication technology. The platform is extensible, so that new device instances can be integrated to it in an efficient and dynamic way, without requiring a reboot of the system. Discovery of new devices is performed automatically, and depending on the type of the detected device, the pervasive platform checks whether it can find the respective control software in the drivers repositories it has access to. If a driver prescribing the functionalities provided by the discovered device can be found, then no manual configuration is needed, otherwise an appropriate description of the new device type has to be added into the system.

The layer is based on an asynchronous publish and subscribe architecture, so that interested parties are notified about the appearance and disappearance of services, as well as about state changes. Several clients can connect to the pervasive layer, such as a Business Process Execution Language (BPEL) engine [Lazovik et al. 2009], a context-awareness component (see Section 3), a visualization software tool (see Section 4), or a user interface interface [Aloise et al. 2011]. These clients subscribe to event types they are interested in, and which concern the change of some state variable. More details about the technologies and standards adopted for satisfying the requirements for the pervasive layer are provided in Section 4.

The Service Composition Layer

Central to the SM4ALL architecture is the composition layer, which is further abstracted into five components. The *repository* keeps the descriptions of the set of supported service types, including appropriate semantic markups about the operations offered, as well as the registry with the actual device instances that are active at any given moment. This is kept up-to-date according to the notifications received from the pervasive layer. A map representing the layout of the house (e.g., the rooms that comprise it, and how they are arranged) is also stored in the repository. Whenever a new device registers itself to the pervasive layer, it also publishes itself to the repository as an instance of its associated abstract type, specifying its functionalities in terms of action *preconditions* and *effects* (see Section 3 for more details). The *context-awareness* component seamlessly monitors the status of the devices and the users' location, collects and aggregates information, and notifies the interested parties via a publish-subscribe mechanism. The *rule engine* uses information about context changes and takes action by directly invoking the composition module, if certain conditions hold (e.g., a fire is detected, and an emergency plan should be put into practice).

The *composition* module receives high-level complex goals issued either by the user layer (e.g., a request for a beer) or the rule engine (e.g., an emergency goal for combating some dangerous gas), and tries to fulfill them by generating appropriate compositions of the available services. The compositions are computed automatically and on-the-fly by a domain-independent planner, which uses constraint satisfaction techniques based on the current home domain delivered by the repository, and the state of the environment provided by the context-awareness module. Given that the working of the composition module relies on the use of AI planning techniques we shall refer to it also as the *planner*. Whenever a goal is issued, the planner generates a plan, that is, a sequence of service operations (actions), whose execution changes the state of the environment in accordance with the properties prescribed by the goal. The plan is then passed to the *orchestrator*, which translates the composition into lower-level service invocations and executes them step-by-step, in a synchronous manner. In case a service operation returns a permanent failure, the plan execution is terminated, the erroneous service is removed from the registry of currently active devices, and the composition module is asked to compute a new alternative plan for the same goal.

The User Layer

The user layer provides the means for the final users to interact with the middleware and instruct the home. The basic module of the user layer is the Abstract Adaptive Interface (AAI) [Catarci et al. 2011], which acts as a proxy that provides services to the particular user interface. Through a unique adaptable algorithm, the AAI is able to manage many different user interface models, such as a touch screen or a BCI, by changing its behavior on the basis of the concrete UI characteristics.

The AAI collects information about the available service operations of active devices and the goals kept in the repository, and forwards them to the concrete UIs. The

information collected from the repository includes visual data (icons) associated with the service operations offered by the devices, as well as information about their location, so that they can be organized accordingly, depending on the capabilities of the concrete UIs. Moreover, a set of icons representing complex goals, such as preparing the bedroom for sleeping, are also made available. The AAI is seamlessly updated to reflect the most recent status of the devices as delivered by the context-awareness component, and notifies the concrete UIs connected to the system accordingly. Whenever an icon is selected, the respective instruction is sent either directly to the orchestrator, if it represents a single operation, or to the composition module, if it corresponds to a complex goal.

3. A CLOSER LOOK AT SERVICE COMPOSITION

By adopting an AI planning approach, the services are synthesized in an automatic and on-demand way, relying solely on individual descriptions of the loosely coupled services exposed by the lower-level available devices. This way, the house provides a high degree of adaptability, rather than being restricted to the support of some predefined tasks. The core component of the composition layer is a domain-independent planner which relies on modeling the domain as a CSP.

3.1. Service Composition via AI Planning

The planner takes the following ingredients as input.

—The representation of the home is in the form of a *planning domain*, that is, the description of the available service operations as *actions*, in terms of preconditions and effects on a set of variables. The planning domain is stored in the repository, together with a description of the home's layout, which specifies the relations between locations (for example, the fact that a room is adjacent to another via a specific door).
—The description of the current state of the home constitutes the *initial state*. The state comprises the current values of all variables that are involved in the home domain, as delivered by the context-awareness module.
—A *goal* prescribes a set of properties to be achieved. The goal can come either from a direct interaction with the user (e.g., "I want to get a beer"), or from the rule engine which identifies whether the current context calls for an automatic intervention (e.g., if the kitchen has a high temperature and the fire alarm is active, then it is likely that there is a fire, and a plan to take care of that should be composed).

Given a goal and the description of a domain instance, the planner first prunes from its search space the actions about which it knows in advance that are irrelevant to the goal, that is, have no potential to contribute to its satisfaction. This preliminary process finds all actions that include at least one of the goal variables in their effects, and then recursively identifies all actions that have as part of their effects variables that are involved in the preconditions of these actions that are directly related to the goal. The search for applicable actions is thus limited to this set of possible candidates. This is a step forward towards avoiding situations where, for instance, the window in the bathroom is opened, while the goal refers to watching TV. Then, the actual plan is generated and passed further to the orchestrator, whose job is to map the plan to a sequence of operations that are provided by the pervasive layer, and to actually execute them.

The planner is informed about any contextual changes it is interested in by the context module, which receives notifications about any changes regarding the involved state variables. Depending on the kind of change that has been identified, a predefined goal may be triggered automatically, for example, when the event that indicates a smoke leak in the kitchen is published. By employing such a public-subscribe mechanism,

the planner can build a plan starting from an initial state for which it has complete knowledge. This holds under the assumptions of continual sensing and information persistence, which imply that the sensors take their measurements and publish the corresponding notifications on frequent time frames, and that the acquired information remains valid until all actions counting on this information are executed.

The planner is equipped with a number of special features that go beyond classical planning [Ghallab et al. 2004] and are of particular importance to the requirements associated with modeling and controlling service interactions in a smart home environment. First, it supports efficient handling of variables ranging over large domains, which are commonly used by smart home components: temperature measurements, TV channels, the number of available items in a smart fridge, the locations monitored by the location component are all essential pieces of information which cannot be efficiently dealt with by traditional planners relying on booleanized encodings.

The representation of the home domain as a dynamic constraint network, which allows connecting and disconnecting constraints on-the-fly, enables the efficient update of the current environmental state as delivered by the context-awareness model, without the need of reloading the whole domain each time a new goal is issued. Whenever an event reporting a change in the home is received by the composition component, a constraint reflecting the new state is incorporated in the CSP, after removing the obsolete information. This mechanism of dynamic addition and removal of constraints is also useful for the interleaving between planning and execution, if we want to also address contingency handling and resolve contradictions that may arise at runtime as shown in Kaldeli et al. [2011]. For instance, recalling the goal of getting a cold beer, let's assume that while the robot is in the storage room and is about to move to the fridge to cool the beer it has got, it finds out that in the meantime the door leading to the kitchen has been locked. In this case, the execution of the plan will fail, and the framework has to resort to the planner again to compute an alternative plan (e.g., by automatically unlocking the door), that can fulfill the goal under the light of the new facts, and given the steps towards the goal satisfaction achieved so far. Although this aspect is not addressed in the current framework, a constraint-based system can be extended to accommodate for intelligent repairing instead of computing a new solution from scratch.

Another important characteristic of the planner, which makes it particularly well-suited for adaptable and user-centric environments, is that it accommodates for a high-level language for expressing extended goals [Kaldeli et al. 2009]. The nature of the goal language is declarative, decoupled from the procedural and operational details of the services. The domain designer or the experienced user who wants to specify a complex request has only to prescribe what properties should be satisfied, without having to know how these can be achieved by the available services. The service operations that fulfill these properties are synthesized by the planner automatically on-the-fly. Temporal aspects, maintainability properties, and distinguishing between a wish to observe the environment or change it are some of the features this language supports.

3.2. Representing the Home as a Planning Domain

3.2.1. The OSGi UPnP-Level Home Domain. All devices that participate in the home domain must have an interface description in accordance with the OSGi UPnP device service specification, so that they can be automatically discovered by the base driver and added to the OSGi registry. Each device exposes its functionalities as one or more UPnP services, which provide a collection of method calls that constitute the *UPnP actions*, and is associated with a set of public variables, called *state variables*. State variables are typed, and can be posted as events, which means that a notification will be generated whenever their value changes. A UPnP action can have multiple

input and output arguments, which according to the OSGi UPnP specification are also represented as state variables. An action may have access to state variables that are associated to other services, and may perform computations on them or actively change them, for example, a robot may be able to manually control external devices. UPnP actions can be distinguished into sensors, which just sense the value of a state variable, and actuators, which change the value of one or more state variables. The home domain can thus ultimately be conceived as a set of UPnP actions which belong to several UPnP services, that in turn are provided by UPnP devices, and can be defined at this low level of the UPnP hierarchy as follows.

Definition 3.1 (*Home Domain*). A home domain (at the UPnP level) is a tuple $\mathcal{HD} = \langle UVar, UPar, USetAct, UGetAct \rangle$ where:

—$UVar$ is a set of variables that reflect some attribute of a service. Each $v \in UVar$ ranges over a finite domain D^v.
—$UPar$ is a set of variables that play the role of input arguments to actions. Each $p \in UPar$ ranges over a finite domain D^p.
—$USetAct$ are UPnP actions that change the value of one or more variables and $UGetAct$ are purely sensing actions that return the value of a variable. We assume that there is a sensing action for every variable of interest $v \in UVar$. These two sets form together the set of all available UPnP actions $UAct = USetAct \cup UGetAct$. Each $ua \in UAct$ has an identifier $id(ua)$ and optionally a set of input arguments $in(ua) \in UPar$. The identifier of the action has the form $id(ua) = DeviceId:ServiceId:ActionId$, where $DeviceId$ and $ServiceId$ are the identifiers of the device and the respective service to which the action belongs. Each device is assigned a unique identifier.

The OSGi UPnP actions describe in a syntactic way the operations that can be performed on the state variables. For example, the OSGi UPnP action "CloseCurtains" sets the value of the Boolean state variable "Curtains" to false. Usually, at this level, the description of the way actions perform is rather primitive, and does not include any checks about conditions that must hold for the action to be invoked in a safe and correct way. For example, given a window that opens inwards, if the "CloseCurtains" action is invoked while the window is open, its casements will interfere with the curtains. Similarly, the action for setting the TV channel will fail its goal if the TV is off, or the action that is responsible for moving the robot may lead to an unfortunate situation if it is performed towards a closed door. This higher degree of reasoning, which is essential for coordinating more complex tasks, is captured by the planning-level semantics.

3.2.2. The Planning-Level Home Domain. In order to automate the task of composition, the OSGi UPnP services have to be enriched with additional semantic annotations which are necessary for the formalization of the available activities, as well as the description of the goal that has to be fulfilled upon a user request or upon a triggering event. To this end, the service operations must be annotated by the domain designer with appropriate semantic markups that capture their functionality in terms of preconditions, modeling the propositions that have to hold in the current state for an activity to be executed, and effects, which formulate how variables are changed by the activity's execution. This set of semantically annotated activities constitute the actions that form the planning domain, which is formally defined as follows.

Definition 3.2 (*Planning Home Domain*). Given a $\mathcal{HD} = \langle UVar, UPar, UGetAct, USetAct \rangle$, a home planning domain is a tuple $\mathcal{PHD} = \langle Var, Par, Act \rangle$, where:
—$Var = UVar$.
—$Par = UPar$.

—*Act* is the set of actions. For each $ua \in USetAct$, there is an action $a \in Act$ which describes its functionality in the following format: $a = (id(a), in(a), precond(a), effects(a))$, where
- $id(a) = id(ua)$;
- $in(a) = in(ua)$;
- *precond(a)* is a propositional formula over $Var \cup Par$ which conforms to the following syntax:

$$precond(a) ::= prop \mid precond(a) \wedge precond(a) \mid precond(a) \vee precond(a) \mid \neg precond(a)$$
$$prop ::= var \circ val \mid var_1 \circ var_2 \mid (var_1 \diamond var_2) \circ val \mid known(var) \mid brel(var_1, \ldots, var_n)$$

  where $var, var_1, \ldots, var_n \in (Var \cup in(a))$, val is some constant, $\circ$ is a relational operator ($\circ \in \{=, <, >, \neq, \leq, \geq\}$), $\diamond$ a binary operator ($\diamond \in \{+, -\}$), $known(var)$ a boolean relation indicating that var is known, and $brel$ an n-ary Boolean relation. We write $\bigwedge_i precond_i(a)$ to denote a sequence of conjunctions on preconditions, and likewise $\bigvee_i precond_i(a)$ for disjunctions.
- *effect(a)* is a conjunction of any of the following elements:
  - $assign(var, v)$, where v is some constant or $v \in Var$;
  - $assign(var, f(v_1, v_2))$, where $v_1, v_2 \in (Var \cup in(a))$ or v_1, v_2 are constants, and f the sum or the subtract function;
  - $increase(var, v)$ or $decrease(var, v)$, where $v \in Var \cup in(a)$ or v is some constant;
  - $cond\_effect(prop, effect(a))$, which models a conditional effect, that is applied at the next state only if *prop* holds at the current state.

It should be noted that the set of sensing actions $UGetAct$ are not represented as planning actions, since their values are updated upon the receipt of the events that are continuously generated by device state changes or by the available sensors as shown in Section 3.2.4. A planning state s is defined as a relation $s = \{(x, D_s^x) \mid \forall x \in Var \cup Par\}$, where $D_s^x \subseteq D^x$, where D^x is the domain of x. The domain of x at state s is given by the *state-variable* function $[\![x]\!](s)$, so that $[\![x]\!](s) = D_s^x$ if $(x, D_s^x) \in s$. If $|D_s^x| = 1$, this means that x at s has a specific value. An action a is applicable on state s if its preconditions hold at s, and its execution leads to a successor state s'. The propositions in $precond(a)$ refer to the values of variables Var and parameters Par at state s, whereas the updates instructed by $effects(a)$ refer to the variables Var at state s'. The domain modeling is based on the multivalued planning task encoding [Helmert 2009], which leads to a smaller number of variables ranging over larger domains, and is particularly well-suited for constraint solvers.

Figure 2 illustrates three examples of home actions with the associated preconditions and effects. The first action $set\_TVChannel(channel)$ states that it can be applied if the TV is *ON* in the current state, and has as a result that the TV is set to *channel*, as provided by the input parameter, in the next state. The second action $close\_bedrCurtains$ refers to opening the curtains of the bedroom window. Because this window opens inwards, the action has as a precondition that the window should be closed. The action $moveRobot(robotLocPar, robotRoomPar)$ instructs how the robot can move to a destination location, provided by the parameter $robotLocPar$, while $robotRoomPar$ refers to the room to which $robotLocPar$ belongs. The action can be applied if $robotLocPar$ does not coincide with the robot's current location, and if either the current and the destination locations are adjacent to each other and belong to the same room, or, in the case they are neighbor locations but in different rooms, the door between these two rooms is open. Other actions, such as opening doors, can be applied to satisfy the preconditions of the *moveRobot* action. This way, the moving will

set_TVChannel(channelPar)
prec: $TV = ON$
Effects: $assign(TvChannel, channelPar)$

close_bedrCurtains
prec: $bedrCurtains = OPEN \ \wedge \ bedrWindow = CLOSED$
effects: $assign(bedrCurtains, CLOSED)$

moveRobot(robotLocPar, robotRoomPar)
prec: $robotLoc \neq robotLocPar \wedge$
$(adjacent\_same\_room(robotLoc, robotLocPar) \vee$
$(adjacent\_diff\_rooms(robotLoc, robotLocPar) \wedge$
$door\_open(robotRoom, robotRoomPar)))$
effects: $robotLoc := robotLocPar$

Fig. 2. Action description examples.

take place in steps, with the robot maneuvering between neighbor locations, based on how these are arranged in the specific house instance. Abiding by such a generic and loosely coupled encoding, the actions that are common in all houses have to be specified once, without being tied to the details of each specific house.

3.2.3. Encoding the Domain into a CSP. A constraint satisfaction problem is a triple $CSP = \langle X, \mathcal{D}, \mathcal{C} \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of variables, $\mathcal{D} = \{D^1, \ldots, D^n\}$ is the set of finite domains of the variables in X so that $x_i \in D^i$, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a finite set of constraints over the variables in X. A constraint c_i involving some subset of variables in X is a proposition that restricts the allowable values of its variables. A solution to a CSP $\langle X, \mathcal{D}, \mathcal{C} \rangle$ is an assignment of values to the variables in X $\{x_1 = v_1, \ldots, x_n = v_n\}$, with $v_i \in D^i$, that satisfies all constraints in $\mathcal{C}$.

Following a common practice in many planning approaches, we consider a *bounded* planning problem, that is, we restrict our target to finding a plan of length at most k, for increasing values of k. Considering a planning home domain $\mathcal{PHD} = \langle Var, Par, Act \rangle$, the target is to encode $\mathcal{PHD}$ into a $CSP = \langle X, \mathcal{D}, \mathcal{C} \rangle$. First, for each variable $v \in Var \cup Par$ ranging over D^x, and for each $0 \leq i \leq k$, we define a CSP variable $x[i]$ in CSP with domain D^x. Actions are also represented as variables: for each action $a \in Act$ and for each $0 \leq i \leq k-1$, a boolean variable $a[i]$ is defined. This way the computed plan can include parallel actions. After deriving the CSP state variables X, the actions' preconditions and effects are encoded into constraints, as explained in Kaldeli [2009a]. Frame axiom constraints are also generated, which guarantee that variables cannot change between subsequent states unless some action that affects them takes place. If some action a_1 affects a variable that is part of the preconditions of some other action a_2, or if both affect the same variable, then a_1 and a_2 are prevented from being put in parallel by an additional constraint.

3.2.4. Incorporating Context Changes. The current value of a state variable may become known either asynchronously via a change event that originates from the invocation of some actuator kind of UPnP action ($USetAct$), or synchronously from the call of some UPnP action of sensing type ($UGetAct$). A sensing action is usually called internally by the respective sensor device, either periodically or when a specific condition in the environment is detected, depending on the type of sensor. It may be also called by any external client that can control the sensor device. A state variable event change or an output argument conveys a tuple *(v, value)*, where $v \in UVar$ and $value \in D^v$. The

new knowledge contained in the tuple is incorporated as a constraint into the CSP as follows.

—When bootstrapping, all sensing actions that are accessible by the orchestrator component are called. Thus, for each variable $v \in Var$ which can be sensed, the retrieved pairs *(v, inValue)* are kept in a structure $mapVarVal$.
—For each v that is included in the bootstrapping phase, a constraint $v[0] = inValue$ is added to the *CSP*, reflecting the current knowledge at the initial planning state.
—Whenever the context-awareness component receives a change event, or the orchestrator calls a sensing action, the respective tuple *(v, value)* is processed: if v is included in the *mapVarVal* structure, and has a current value $inValue$ and $inValue \neq value$, then the constraint $v[0] = inValue$ is removed from the *CSP*, and the constraint $v[0] = value$ is added.

Besides changes in the values of variables, a contextual change may reflect the realization that a service has become unavailable, if the response after a synchronous call of the UPnP action *ua* by the orchestrator indicates a permanent failure. In such a case, the semantic repository is notified that the respective operation is not functioning properly anymore, and removes it from the registry of available services. The repository thus publishes an event which indicates that the action *ua* has become unavailable, and in turn, the following constraints are added to the *CSP*: for all $0 \leq i < k$, $a[i] = false$, where $a[i]$ is the CSP-level boolean variable modeling the planning action that corresponds to the UPnP action *ua*, with $id(a) = id(ua)$. This way, subsequent plans are not allowed to include action a in any step. If the services become available again, then the preceding constraints are dynamically removed from the CSP, upon the appropriate notification received from the repository. We remark that the connection and disconnection of constraints is postponed if the constraint solver is currently searching for a valid assignment. Therefore, under certain circumstances, the solution plan computed by the solver may rely on assumptions about the contextual state that have become out-of-date.

3.2.5. Extended Goals. A set of predefined goals depending on the user's routine and needs are made available through the set of buttons modeling complex activities that appear in the control panel of the supported UI. If the goal issued can be satisfied, the generated plan is executed, and the home devices change state accordingly. If the goal is not satisfiable under the current context, a message is shown on the user interface. Table I shows how the goals described in natural language in the scenarios of Section 1.3 are expressed with respect to the goal language supported by the planner. A detailed description of the goal language and its formal semantics can be found in Kaldeli [2009b]. An `achieve-maint`$(\wedge_i prop_i)$ subgoal on a conjunction of propositions $prop_i$ on some domain variables implies that $\wedge_i prop_i$ has to become true at some state, and stay true until the final state of the produced plan. The `find_out` type of subgoals take care of sensing. In the case of `achieve-final` subgoals, the respective proposition has to be satisfied at the final state, but is allowed to hold or not throughout the plan execution, like, for example, in Goal 5 where *robotLocation* will change many times while the robot is moving around to find and get the beers. The construct $goal_1$ `under_cond` $goal_0$, for example, used in Goal 3, instructs that first the planner will pursue the satisfaction of $goal_0$, and then the satisfaction of $goal_1$. If $goal_0$ is unsatisfiable, then the goal will fail. Thus in the case of Goal 4 for moving to the kitchen if the smoke there has been eliminated, only if $kitchSmoke = OFF$ holds will the rest of the goal be carried on. In contrast, a goal of the form $goal_1$ `under_cond_or_not` will also be fulfilled if $goal_1$ is not satisfiable, if, however, it is, then $goal_1$ has to be as well. For example, in Goal 2 this structure ensures that the subgoal $sitrAirCond = ON$ will be satisfied if the

Table I. Goals for the Smart Home

Goal 1: wake up (by Rule Engine)	achieve-maint ($alarmClock = ON \ \wedge \ bedrCurtains = OPEN$ $\wedge \ bedLevel = HIGH$) $\wedge$ achieve-maint ($bedrLight = ON$) under_cond_or_not find_out-maint ($bedrNatLight = LOW$)
Goal 2: watch TV (by UI)	achieve-maint($TvChannel = CH5 \ \wedge \ personRoom = SITR$) $\wedge \ sitrLight = MEDIUM$) $\wedge$ achieve-maint($sitrCurtains = CLOSED$) under_cond_or_not find_out-maint($sitrNatLight = LIGHT$) $\wedge$ achieve-maint($sitrAirCond = ON$) under_cond_or_not find_out-maint($sitrTemperature > 30$)
Goal 3: deal with smoke leak (by Rule engine)	achieve-maint($kitchVentilator = ON \wedge$ $TvState = ALARM \ \wedge kitchWindow = OPEN$) $\wedge$ achieve-final($doorsLeadTo(KITCHEN) = CLOSED$) under_cond_or_not achieve-maint($personRoom \neq KITCHEN$)
Goal 4: smoke eliminated (by Rule engine)	achieve-maint($kitchVentilator = OFF \ \wedge \ TV = OFF$)
Goal 5 go to kitchen if safe (by UI)	achieve-maint($userLocation = AT\_OVEN$) under_cond find_out-maint($kitchSmoke = OFF$)
Goal 6: bring cold beer (by UI)	achieve-final($robotLocation = userLocation \wedge$ $robotHolds = BEER \ \wedge \ beerTaken = COLD$)
Goal 7: health emergency (by Rule Engine)	achieve-maint($nurseNotif = healthEm + \text{'\_}ALARM\text{'} \ \wedge$ $hospitalNotif = healthEm + \text{'\_}ALARM\text{'} \ \wedge$ $robotLocation = userLocation$) under_cond_or_not find_out-maint($nurseLocation = OUTSIDE$)
Goal 8: go to sleep (by UI)	achieve-maint($bedLevel = LOW \ \wedge \ alarmClockTime = 08{:}00$ $bedrCurtains = CLOSED \ \wedge \ bedrLight = OFF$)

temperature is higher than 30 degrees, while if the temperature is lower than that, then only the rest of the conjunctions of subgoals will be looked after. Note that in the case of Goal 8, $healthEm + \text{'\_}ALARM\text{'}$ refers to a concatenation of strings, taken care by an external method call.

As becomes obvious by the provided examples, the user doesn't have to know about the operational details of the service instances available in each specific house. It is up to the planner to find a "smart" solution based on the capabilities of the particular house and the current context, without depending on any ad hoc business processes. Thus, given a goal, the composition module may come up with completely different plans, depending on the domain instance and initial state. Moreover, the cause that triggers an event can also be taken into account: in the case of the health emergency goal, the notification message sent to the nurse's mobile phone and to the hospital incorporates the cause of the failure. If the rule engine triggered the goal because it recognized a fall, then $healthEm = \text{'}FALL\text{'}$, if the context conditions indicated a heart attack then $healthEm = \text{'}HEART\_ATTACK\text{'}$, etc. The heath emergency recognition can be based on complex computations on several sensed context variables, like, for example, presented in Li et al. [2009]. For example, a fall could be detected based on the measurements delivered by wearable sensors, such as gyroscopes and accelerators, for example, $angle_2 \div angle_1 > v1 \ \wedge \ acc_x > v2 \ \wedge \ acc_y > v3 \rightarrow healthEm = \text{'}FALL\text{'}$.

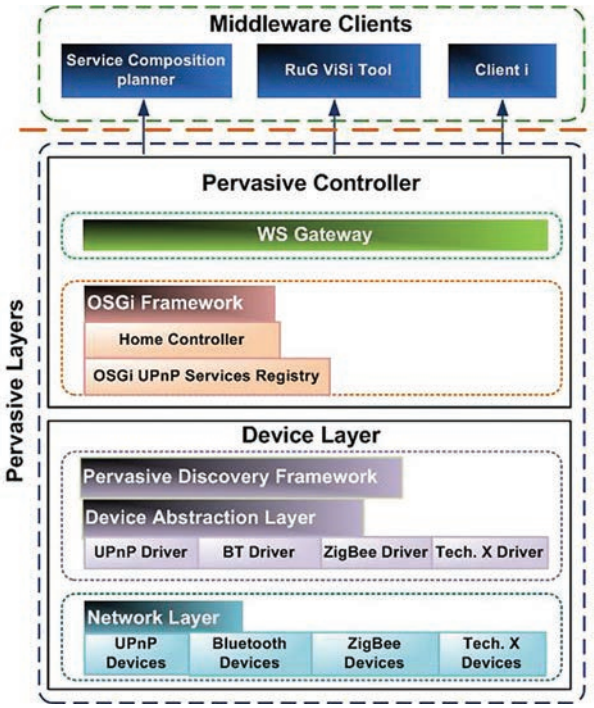

Fig. 3. Architecture of the pervasive layer.

In Section 6, we present the plans generated by the planner module for each of the goals in Table I for a specific smart home domain and for the particular initial states we have used for testing our scenarios. Invariant constraints, stating some conditions that should hold at the initial and final state of every plan, or should never be violated at any state, may also be added by the domain designer.

4. THE PROTOTYPE

The SM4ALL architecture is fully implemented to test its technical properties, but also the experience of users with it. Next we illustrate the prototype built based on the design presented in Section 2 and proceed by architectural layers.

4.1. Pervasive and Composition Layers

We use the UPnP protocol to control the hardware devices, HTTP to enable access to remote clients, and the OSGi service platform as the intermediary between the physical UPnP layer and the service endpoints. The implementation is based on the Apache Felix project[4], which is a framework for writing devices exposed as UPnP (conforming with the OSGi UPnP specification version 1.1) and integrating them into the OSGi bundle repository. The interface of the services is written in Java. Figure 3 provides an overview of the internal structure of the pervasive layer, and the standards it uses. At the bottom sits the *network layer*, where physical devices with different networking protocols are located. The *device abstraction* layer abstracts away the underlying device technology by offering a driver for each of the technologies that the pervasive middleware supports.

[4]felix.apache.org.

UPnP is used as the device-neutral technology to which all devices are wrapped by the respective driver, so that they can be then registered as OSGi services.

Besides UPnP, the prototype is able to automatically discover and support Bluetooth and ZigBee[5] devices, but it can be easily extended by adding drivers for other technologies as well. All devices' provided functionalities, independently of their network protocol, are described in compliance with the format prescribed by the OSGi UPnP specification, based on two types of elements: actions, which describe the operations a service supports, and *state variables* which represent the current state of an UPnP service. Whenever the value of a state variable changes, the respective event is published and propagated to the upper layers, notifying all subscribed parties.

OSGi is used as the platform to expose the devices' functionalities as services to the application layers. All components participating in the OSGi framework are deployed as so-called "bundles". The *controller* is a special OSGi bundle that is responsible for handling events and controlling the services available in the framework, functioning as a bridge between the OSGi layer and the *WS gateway*, which executes a lightweight HTTP server that provides a standardized API to external components. Several clients can be registered to the server running on top of the OSGi framework, and call the exposed operations, such as getting the list of available services, subscribing to state variable events, or invoking an action offered by a service. Clients can be a BPEL engine, a home visualization application [Warriach et al. 2010], or the SM4ALL orchestrator component.

The context-awareness module is registered as a client to the WS endpoint on top of the pervasive layer, and subscribes to all change events of the variables involved in the service descriptions. The orchestrator is also a client to the pervasive layer, without, however, subscribing to any variable change events: all it has to do is to be able to invoke services through the respective operation exposed by the WS server, as instructed by the composition component or directly by a simple command coming from the user layer. The semantic repository is yet another client of the pervasive layer, which is notified about the registration and deregistration of services, so that it adds a new instance of the abstract description of the associated service type. The pervasive layer and the clients registered to it interact through the exchange of XML messages.

As already mentioned, the planner translates the semantic domain representation into a CSP. The context awareness acts as a listener to UPnP change variable events, which are further processed by the planner so that they are incorporated at the initial state of the evolving CSP. Whenever a goal is issued, a standard constraint solver is applied to the current instance of the CSP to find a solution which amounts to a valid plan. We use the Choco v2.1.1 constraint programming library[6], which provides a large choice of implemented constraints, as well as a variety of predefined but also custom search methods [Kaldeli et al. 2011]. The invocations of operations by the orchestrator take place in a synchronous way, so that the next action in a totally ordered plan is invoked after a success return value is received by the previous action invocation.

4.2. The User Layer

The Abstract Adaptive Interface (AAI) [Catarci et al. 2011] is registered as a client of the server on top of the OSGi framework, and whatever commands are issued via the concrete UIs are passed through it to the lower levels of the architecture. Its implementation is based on Apache Tomcat and Apache Axis. Two kind of UIs, a standard Web-based and a Brain Computer Interface, are connected to the AAI proxy, with which they interact via the exchange of XML messages. The Web interface (or

[5]www.zigbee.org.

[6]www.emn.fr/z-info/choco-solver.

Fig. 4. Views available via the Web interface.

alternatively referred to as control panel) consists of dynamic and responsive web pages developed in JSP (Java Server Pages). The web pages provide different views of the virtual home. A global view presents all commands available to the user, in the form of clickable icons along with some descriptive text, corresponding either to atomic service operations, such as "turn on the light in the bedroom", or to complex goals (discussed in the following section) stored in the repository. The icons represent the current state of the devices, for example, the icon indicating "Kitchen light ON" reflects the fact that the current state of the respective light is on, and clicking on it entails turning the light off, and thus refreshing the web page accordingly, after the notification originated by the respective UPnP device is received. A page depicting the rooms of a virtual apartment allows the user to move to the view of a particular room, from which only the devices whose location matches this room can be seen and controlled, as depicted in Figure 4. An extra Web page is reserved to reflect the current state of the devices at the user layer when the BCI is used.

The BCI [Guger et al. 2009] is intended for users who have lost part of their motor ability due to aging or chronic neurological disorders, and are therefore unable or find it difficult to control the system via the standard Web interface. The BCI used in the test sessions is a portable asynchronous P300-based one, which translates the users' voluntary ElectroEncephaloGraphic (EEG) modulations into a control signal sent to some external device. The set of available commands modeling devices and goals are presented on a computer monitor in a form of a 4 by 4 matrix of flashing icons, which are flashing in a random order. The user wearing the EEG cap has to concentrate on a specific symbol, and whenever this is highlighted, a particular component is recognized in the measured EEG data. As a result, the identified command is transmitted to the AAI proxy. Because the flashing icons have to be static, that is, they represent a device rather than its current state, the effect of the commands issued through the BCI are reflected via the Web page of the Web interface reserved for this purpose, which is updated whenever the state of a device changes. Figure 5 shows a user wearing an EEG cap, having the list of the screen with the flashing icons-commands on her left side, and viewing a projected simulation of a smart home. More details about the technology and testing results concerning the BCI can be found in Aloise et al. [2010].

Fig. 5. A BCI user interacting with the virtual home.

4.3. Simulation and Visualization

Setting up an actual physical home or lab facility, furnished with modern sensors and actuators, is particularly expensive and effort-demanding, and performing tests with end-users in it can be inefficient. Therefore, it is instrumental to be able to acquire feedback from users before moving to the actual home and installing the real hardware devices, so that their requirements are taken into account early in the development process. To this end, we have implemented a virtual home environment which mimics as closely as possible an actual home setting, with simulated home services substituting physical hardware.

The implementation of the simulation and visualization platform (the RuG ViSi tool) is based on Google SketchUp and has been demonstrated [Lazovik et al. 2009; Warriach et al. 2010]. It is integrated in the framework as a client of the WS endpoints at the pervasive layer. The apartment modeled is equipped with virtual devices implemented in Ruby[7], which are coupled with the Web services exposed by the devices in the pervasive layer. In this way, whenever a device state is changed, the result is replicated in real time in the visualized home. For instance, to model a reaction to fall detection, we have coupled a virtual alarm in the simulated house with a Sentilla mote[8] equipped with an accelerometer. The device uses the ZigBee communication protocol, and is wrapped in the OGSi layer. When shaken, the virtual alarm is turned to red, indicating a warning about the fall. The position of a user in the house is also shown, by coupling a user virtual service with a location detector that provides information about the user's location. Conversely, one can also control the devices at the pervasive layer through their virtual equivalents, so that there is a one-to-one mapping between the state of the OSGi UPnP-level environment and its visual reproduction. Figure 6 depicts a screenshot of a virtual house, and shows how visualized devices at the RuG Visi level, such as lamps or the TV, interact with OSGi UPnP devices. In case of a composition, the series of effects entailed by the orchestrated UPnP-level operations are reflected in the appropriate sequence at the visualization level.

4.4. Sample Interaction Flow

In order to demonstrate how the different components of the SM4ALL prototype are integrated and cooperate with each other, we go through a simple scenario and describe the control and information flow which realizes the desired behavior. Let us consider

[7]www.ruby-lang.org.
[8]www.sentilla.com.

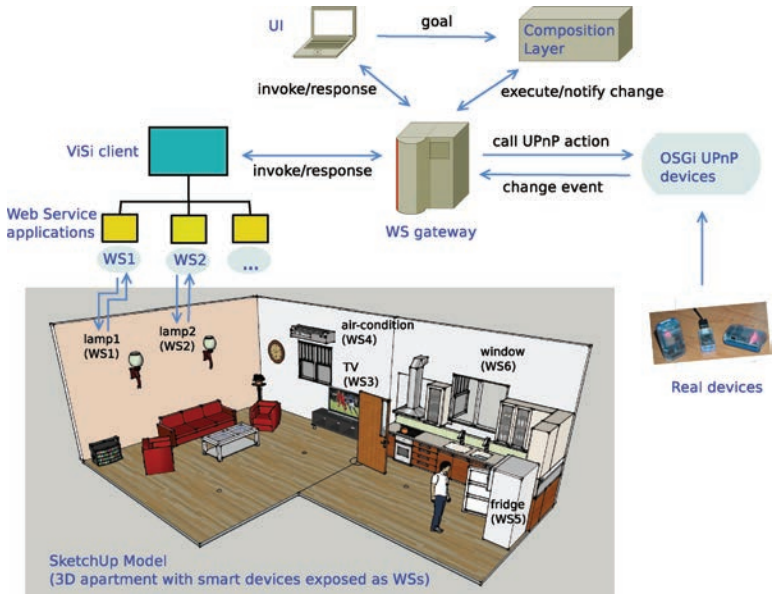

Fig. 6. A screenshot of the home simulation output.

an example with a single physical device of type *Lamp*. The description of the *Lamp* type includes one published boolean-valued UPnP variable, *status*, and three UPnP actions, *turnOn*, *turnOff*, and a sensing one that returns the current value of *status*. It is also annotated by an appropriate semantic representation of the two actuator operations in terms of preconditions and effects. This description is stored in the semantic repository, as an XML file. During bootstrapping, the device is automatically discovered thanks to the OSGi UPnP platform, and the semantic repository is notified about its subscription. As an effect, the semantic repository produces an instance-specific semantic description of the operations offered by the particular device, by adding the device's unique identifier (*lamp1*) as a prefix to the variables and actions that are declared in the abstract type *Lamp* description. All subscribed components are notified about the lamp availability and its description, so that based on that, the planner produces a domain consisting of one variable (*lamp1*::*status*) and two planning actions (*lamp1*::*turnOn* and *lamp1*::*turnOff*). To inform all interested parties about the current state of all devices, the orchestrator invokes all available sensing actions. As an effect, the lamp device publishes an event which contains the current value of *lamp1*::*status*. The context-awareness component forwards this event to the UI, the rule engine, and the planner, which sets accordingly its initial state (see Section 3.2.4).

Let us also assume that a trivial goal, which specifies that *lamp1*::*status* should be *TRUE*, is also stored in the repository. Thus, the UI along with the subscribed services is also notified about the existing goals, and thus presents in the Web interface the appropriate icon. When the user selects this icon, a request for producing a composition that satisfies the respective goal is forwarded by the AAI proxy to the planner. Assuming that at the current state *lamp1*::*status* = *FALSE*, the planner computes a plan consisting of a single action *lamp1*::*turnOn*. The plan is passed to the orchestrator, and ultimately the UPnP action called *turnOn* that belongs to device *lamp1* is called at the pervasive layer. The call is synchronous, and if successfully fulfilled a "success" reply is returned to the orchestrator. Moreover, a change event concerning the *lamp1*::*status* variable of *lamp1* is published by the pervasive layer, and is ultimately received by all

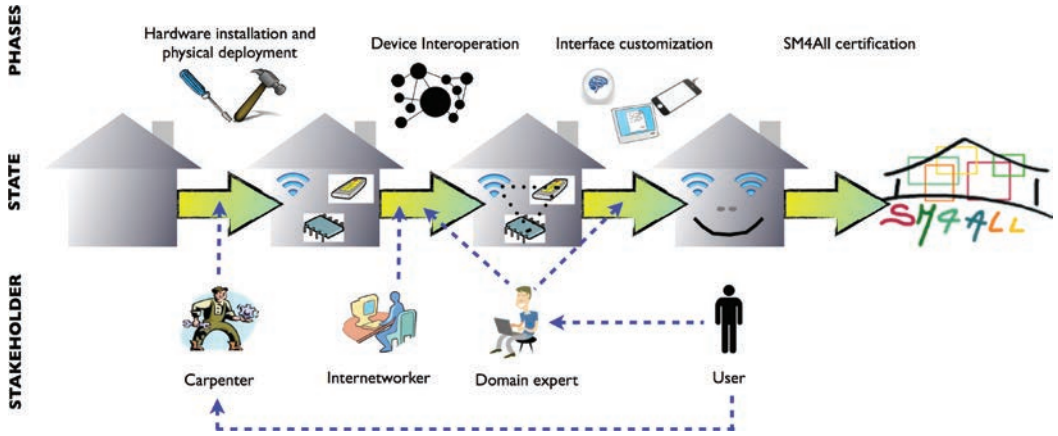

Fig. 7. The process of fitting the SM4ALL architecture into an existing home.

subscribed clients: the planner updates its initial state, so that it reflects the most latest values of the UPnP variables, and similarly the UI changes the icon which indicates the state of the lamp. If the response received by the orchestrator indicates that the lamp is broken, the orchestrator notifies the semantic repository, which takes the initiative to unsubscribe the service from the OSGi UPnP platform, and asynchronously notify about this removal all interested parties. Thus, if the goal for turning on the light is issued again, the planner will respond that no plan can be found.

5. PRACTICALLY ENGINEERING A SMART HOME

The SM4ALL architecture can in principle be fitted to any existing home. In the following, we describe the actual phases that such a fitting process, necessary to make a home smart with respect to the SM4ALL approach, would have to go through. Clearly, the average home user does not want to bother with technical details, and is willing at most to provide some input on what are the goals he wishes to regularly perform in the home. In Figure 7, we provide a schematization of the process where we show from left to right the state in which the home goes through, and we distinguish the engineering phases (top) and the stakeholders who are actually responsible for successfully completing each phase of the process (bottom).

The first phase of the SM4ALL fitting process requires to make an inventory of the devices present in the home and identify which additional hardware is required to cover the user needs, for example, door motors, smart meters, smart fridge, etc. Then the new devices have to be physically installed in the home. In this initial phase, it is mostly to the SM4ALL expert to do the requirement engineering and to the carpenter to fit the hardware in the home. The second phase consists of making the hardware interoperate, which relates to the pervasive layer of the SM4ALL architecture. An internetworking expert has to make sure that all devices are connected to the network and can exchange messages. This is in principle effortless for the devices which adhere to UPnP standards, and should be easily achieved also for other devices based on known protocols.

If a new type of device is introduced into the SM4ALL system, that is, the functionalities it offers have never been described before in terms of preconditions and effects, it is also necessary to add these extra semantics. If the new device, for example, a particular lamp, is an instance of a known service type, for example, the lamp-A type, then all that has to be done is to declare the type of the device. This task is performed by the domain designer. The effort of this stakeholder is thus considerable at the beginning, when the

behaviors of the supported device types have to be semantically specified, and gradually diminish as more and more devices are added to the semantic repository to be used by the composition layer. The last technical phase of the process consists in customizing the interface to the home for the user. This means identifying the appropriate type of interface hardware (e.g., BCI, touch screen, voice interface), and also the complex requests according to the users' needs and routine. The requests are formulated by the domain designer or the experienced user in conformity with the declarative fashion of the extended goal language, and are tied to an icon that appears in the user interface. Emergency goals are also formulated along with the conditions that enact them and added to the rule engine. Again, the specification of the goals requires more effort in the first installations, and becomes less demanding as reuse of already formulated goals becomes the norm. Finally, one could imagine a final certification phase where an expert or standardization body may certify the home to be SM4ALL compliant, thus allowing for interoperation with new SM4ALL certified hardware.

Considering the shift towards interoperable and service-oriented home setting, OSGi UPnP is a good candidate for constituting a common standard for home appliances, especially since it can easily support different network protocols. However, the vendors who offer devices with a ready-to-use OSGi interface, certified as "OSGi compliant", are still limited (these include Samsung, 4DhomeNet, IBM, Connected Systems, and others). Given that the reality in home appliances is still far from the adoption of some common standard, the task of enabling compliance with the OSGi platform falls on the home designer. Depending on the specifications of each device, this task may vary from easy to difficult or impossible. For some frameworks it is possible to wrap them directly as OSGi bundles, for some implementing an adaption layer is necessary, while for others some patching of their sources is unavoidable. To give an example from our own experience, the task of representing a Sentilla accelerometer in terms of the UPnP standards is a matter of less than an hour, however, implementing an adaptor for Zigbee, the network protocol used by the device, required a couple of weeks development time. However, an adaptor for a given network protocol needs to be developed only once, and as long as well-known protocol discovery plug-ins and adapters are available at the OSGi home gateway, new devices that use these protocols can be automatically integrated.

6. TECHNICAL EVALUATION

We provide both a technical evaluation of the system to assess whether the architecture is effective and the used techniques have adequate performance; and a user evaluation in Section 7 to give an initial assessment of the acceptability and usability of the solution. The major focus of the technical evaluation is on the composition component which, given the necessity to work on large search spaces, may raise concerns of inefficiency. The evaluation is an extension of the preliminary one presented in Kaldeli et al. [2010], and is based on the scenario described in the Introduction and detailed in Section 3. The tests have been run on a 1.83 Ghz computer running debian lenny, 32 bit and Java 1.6.0_12. The service components are simulated with accordance to the OSGI UPnP device specification and are exposed as OSGi bundles. Each device offers one or more services, each of which involves a number of actions and state variables.

The composition layer subscribes as a client to the Web service server: the semantic repository gets the list of active devices and provides the respective action descriptions, the context-awareness component subscribes to the events regarding all domain variables, and the orchestrator is also connected, ready to receive invocation instructions. For the evaluation purposes we model a home with 5 rooms, and 14 simulated UPnP devices. For simplicity, we have simulated one aggregated device for managing all doors by passing the specific door which the open/close operations affect as an input argument, thus having one device for controlling 4 doors, and a similar case holds for the lights,

window, and curtains devices. The total number of UPnP actions implemented by the devices is 28 (plus the sensing actions that are defined for each state variable in the domain) and involve 37 different state variables. These actions model the getting and setting of the declared state variables. For example, the air condition device comprises two state variables, $AirConditionState \in Boolean$ and $AirCondTemperature \in Integer$. It also offers two UPnP actions of activator type, each one defined in a separate service: a $SetEnumStateVar$ for turning on and off $AirCondState$, and $SetIntStateVar$ for controlling the desired $AirCondTemperature$. For the purposes of simulation, the user himself is represented as one of the services at the pervasive layer. One can think of a person on a wheel-chair which can be controlled automatically, and its position is being tracked by a localization component. The robot device refers to a robot that, without loss of generality, is dedicated in the testing scenario to the task of bringing beers to his master: it can move around the house, get a beer from the fridge or the storage, sense if it is cold or not, and cool it if necessary by putting it in the fridge and waiting for some time. A state variable can be involved in more than one services, possibly belonging to different devices, like the $FridgeDoor$ variable, which can be controlled automatically or directly by the *Robot* device.

Table II shows the sequence of actions generated by the planner for each of the goals in Table I (because of space issues, only the initial values that are of interest to the goal are mentioned in the table, rather than the full initial context of the home and user). Each plan is represented as a partially ordered set of actions, with comma-separated actions $a1, a2$ indicating that action a_1 has to be performed before a_2, while the actions included in the same set $\{a_1, \ldots, a_n\}$ can be executed in parallel. One can see that depending on the current contextual state, the plans which satisfy a given goal may be radically different. The planner produces plans with a high degree of parallelism, that is, most actions that are independent of each other are put in parallel. However, in the current implementation, the orchestrator does not support parallel action executions, and thus the actual invocations are performed in a serialized manner. Because of the use of random search strategies, the plans returned may slightly vary between different runs: the order of some actions may be different or some extra actions may be included. The latter is due to the fact that the planner does not generate optimal plans, that is, the ones comprising the minimum possible number of actions. Thus, it may occur that a plan includes unnecessary actions or useless repetitions of actions, as, for example, in plan `[5a]`, where some doors are opened for no reason without that being necessary for the goal's satisfaction. The preliminary step for removing the irrelevant actions cannot prevent such situations, because it can only rely on the knowledge it has prior to performing the actual planning. So, it cannot, for example, predict whether the robot will need to move to the storage for getting a beer or not, which makes a difference for the relevance of, for example, the $open\_kitchStorDoor$ action.

The time required by the planner to subscribe to the available UPnP services, build the planning-level domain description, and sense the first initial state, by invoking the UPnP sensing actions for all state variables, is 10.8 sec. After that, it is ready to generate plans for the goals that are issued by the user or the rule engine, while it is notified about any changes by the context module, and updates its current initial state accordingly. We have measured the time the planner takes to compute each of the plans, as well as the time needed for each plan to be actually executed by invoking the respective UPnP actions. These results are summarized in Table III. We have used a random branching strategy during constraint solving, by resting the search after a maximum number of backtracks. The reported times both for composition and execution are the average over 5 separate runs. Of course, if we consider real rather than simulated devices, the execution times especially for motion-related actions would be much longer. The most demanding goal is Goal 6 (getting a cold beer),

Table II.
The plans generated for the goals in table I for different initial states (only the initial values that are of interest to the goal are mentioned).

Initial state	Plan
	Goal 1 (wake up)
[1a]: *bedLevel=LOW, bedrNaturalLight=DARK, bedrCurtains=CLOSED, bedrLight=OFF*	{*set_bedLevel(MEDIUM), ring_alarmClock, open_bedrCurtains turnOn_bedrLight*}, *set_bedLevel(HIGH)*
[1b]: Same as above, but with *bedrNaturalLight=LIGHT*	{*set_bedLevel(MEDIUM), ring_alarmClock, open_bedrCurtains*}, *set_bedLevel(HIGH)*
	Goal 2 (watch TV)
[2a]: *TV=OFF, sitrLight=LIGHT, sitrTemperature=32, sitrNaturalLight=LIGHT, sitrCurtains=OPEN,* $\forall i$ *door*$_i$*=CLOSED*	{*turnOn_lightSitr(MEDIUM), close_sitrCurtains, turnOn_sitrAirCond, set_TV(ON)*}, *set_TVChannel(CH5)*
[2b]: Same as above, but with *sitrTemperature=20, sitrNaturalLight=DARK*	{*turn_lightSitr(MEDIUM), set_TV(ON)*}, *set_TVChannel(CH5)*
	Goal 3 (deal with smoke leak)
[3a]: *userLocation=AT_OVEN, TV=ON, kitchWindow=CLOSED, ventilator=OFF, kitchSitrDoor=OPEN*	{*turn_on_ventilator, open_kitchWindow, open_kitchSitrDoor, set_TV(ALARM)*}, *moveUser_to(AT_KITCH_DOOR), moveUser_to(AT_TV), close_kitchSitrDoor*
[3b]: Same as above with *userLocation=AT_TV*	{*turnOn_ventilator, open_kitchWindow, close_kitchSitrDoor, set_TV(ALARM)*}
	Goal 4 (smoke eliminated)
[4]: *kitchSmoke=OFF, TV=ALARM, kitchVentilator=ON*	{*turnOff_kitchVentilator, set_TV(OFF)*}
	Goal 5 (go to kitchen if safe)
[5a]: *kitchenSmoke=ON*	The goal cannot be satisfied
[5b]: Same as above but with *kitchenSmoke=OFF, userLocation=AT_STOR_ENTR*	*open_kitchStorDoor, moveUser_to(AT_FRIDGE), moveUser_to(AT_OVEN), close_kitchStorDoor*
	Goal 6 (get cold beer)
[6a]: *robotLocation=AT_START, userLocation=AT_SOFA, kitchStorDoor=CLOSED, sitrKitchDoor = CLOSED, numOfBeersInFridge=0, numOfBeersInStorage=8, robotHolds=EMPTY, fridgeDoor=CLOSED*	{*open_sitrKitchDoor, open_kitchStorDoor*}, *moveRobot_to(AT_OVEN), moveRobot_to(AT_STOR_SHELF), robotGetsBeerFromStorage, open_fridgeDoor, moveRobot_to(AT_FRIDGE), robotCoolsBeer,* {*open_fridgeDoor, close_kitchStorDoor*}, *robotGetsBeerFromFridge,* {*moveRobot_to(AT_SOFA), close_fridgeDoor*}
[6b]: Same as above but with *numOfBeersInFridge=1*	*open_sitrKitchDoor, moveRobot_to(AT_OVEN), open_fridgeDoor, moveRobot_to(AT_FRIDGE),* {*robotGetsBeerFromFridge, open_kitchStorDoor, open_bedrBathrDoor, open_sitrBedrDoor*}, {*moveRobot_to(AT_SOFA), close_fridgeDoor*}
	Goal 7 (health emergency)
[7]: *nurseLocation=OUTSIDE, cause=FALL,* $\forall i$ *door*$_i$*=CLOSED, robotLocation=AT_TV*	{*sendMsg_NursePDA(FALL_ALARM), notifyHospital(FALL_ALARM), open_sitrBedrDoor, open_bedrBathrDoor*}, *moveRobot_to(AT_BED), moveRobot_to(AT_BATH)*
	Goal 8 (go to sleep)
[8]: *bedLevel=UP, bedrCurtains=OPEN, bedrWindow=OPEN, bedrLight=ON*	{*set_bedLevel(MEDIUM), close_bedrWindow, set_alarmClock(08:00), turn_on_bedrLight=OFF*}, {*close_bedrCurtains, set_bedLevel(LOW)*}

Table III. Time Required for Composition and Execution

Test	Number of actions in plan	Plan Composition time (in sec)	Plan execution time (in sec)
[1a] (wake up, no natural light)	5	1.1	0.5
[1b] (wake up, natural light)	4	1	0.3
[2a] (watch TV, temperature too high, natural light)	5	1.5	0.7
[2b] (watch TV, temperature ok, no natural light)	3	1.4	0.6
[3a] (deal with smoke, user in kitchen)	7–9	1.2	1
[3b] (deal with smoke, user not in kitchen)	4	0.6	0.4
[4] (smoke eliminated)	2	0.7	0.4
[5a] (go to kitchen, smoke on)	0	0.1	-
[5b] (go to kitchen, smoke off)	4–5	0.7	0.4
[6a] (get beer, fridge empty)	12–15	2.6	0.8
[6b] (get beer, fridge full)	7–10	2.2	0.7
[7] (health emergency)	6	1.4	0.5
[8] (go to sleep, bedrWindow open)	6	1.2	0.6

especially in the case where there are no beers already stored in the fridge, mainly due to the substantial backtracking required to find a solution (up to 478 backtracks, compared to 47 backtracks in the worst case concerning the other goals). The invocation time per operation call is a up to a few msec for all devices, since these are simulated. The execution time amounts to the time required for the interaction between the composition module and the orchestrator, that is, the time for mapping the planning actions to UPnP actions, calling them, and getting the response, while at the same time a listener parses the UPnP change variable events and updates the CSP.

An evaluation of the performance of the pervasive layer with respect to the number of clients it can support in association with the number of connected devices is beyond the scope of the current presentation. Results with respect to such a parameter are presented in Kaldeli et al. [2010].

6.1. Replanning Scenarios

For the purpose of simulating failure handling scenarios, we only consider two basic kinds of UPnP action invocation responses: “success” and “failure”. The policy upon a failure response is first to try to invoke the erroneous operation once more, and if a failure occurs for a second time, then to attempt to replan. The application of different policies depending on the kind and severity of contingencies that are detected during execution may be possible if a more subtle distinction of the cause of failure is available (e.g., attempt to reinvoke the same service several times, or directly remove the service if the response indicates a permanent failure). Timeout conditions may differ depending on the type of action (e.g., a service operation for closing/opening a door should respond within a second, while closing the curtains takes longer). Timeouts are handled the same way as failures.

Table IV summarizes the behavior and performance of the system under certain circumstances concerning the scenarios described in Section 1.3, where an error occurs during the execution of the initial plan. The services used for the tests are the same as before, with the addition of three extra service actions. For the purpose of scenario 1, which refers to the goal for watching TV, a “switchOn” operation is added to the robot device, which models its ability to turn on the TV if its location is in front of it. To

Table IV.
Behavior and time results of the planner for two possible replanning scenarios depending on execution circumstances.

Scenario 1: Re-planning for Goal 2 (watch TV)	
Initial state:	as in TableII[2a] and robotLocation=AT_BED
Initial plan:	as in TableII[2a]
Execute plan:	*set_TV(ON)* responds with failure twice, re-planning
New plan:	*open_sitrBedrDoor, moveRobot_to(AT_SOFA) , moveRobot_to(AT_TV), robotSetTV(ON), set_TVChannel(CH5)*
Execute plan:	Completed successfully
Planning attempts: 2	
Total planning time: 3.3 sec	

Scenario 2: Re-planning for Goal 6 (bring cold beer)	
Initial state:	as in TableII[6b]
Initial plan:	as in TableII[6b]
Execute plan:	*open_sitrKitchDoor* responds with failure twice, re-planning
New plan:	{*open_sitrStorDoor, open_kitchStorDoor*}, *moveRobot_to(AT_STOR_SHELF), open_fridgeDoor, moveRobot_to(AT_FRIDGE), moveRobot_to(AT_SOFA)*
Execute plan:	*open_kitchStorDoor* times out twice, re-planning
New plan:	{*open_sitrBedrDoor, open_bedrKitchDoor*}, *moveRobot_to(AT_BED), open_fridgeDoor, moveRobot_to(AT_FRIDGE), moveRobot_to(AT_SOFA)*
Execute plan:	*open_sitrBedrDoor* times out twice, re-planning
New Plan:	The goal cannot be satisfied
Planning attempts: 4	
Total planning time: 12.3 sec	

simulate the different scenarios regarding scenario 2, two extra door services are added, to simulate the possibilities for the robot to follow alternative routes to reach the kitchen.

In scenario 1, after the invocation to remotely turn on the TV fails, a second attempt is made, and after the service responds with a failure again, the erroneous action is removed from the constraint network, through the addition of the appropriate prohibitive constraint (see Section 3.2.4). The planner is called again, and the new composition instructs the robot to move from the bed where it currently is to the TV and switch it on. In scenario 2, the robot cannot move from the sitting room to the kitchen directly, because the door that connects the two rooms is blocked. After pruning the faulty door from the search space, an plan that leads the robot to the kitchen through the storage room is generated. However, the door that connects the living room with the storage room also proves to be defect, and the invocation for opening it times out. As a result, the planner will try to find an alternative route through the bedroom. Due to bad luck though, it turns out that the door to the bedroom is also out of order, and the planner will make a fourth attempt to compute a plan which does not include any of the malfunctioning doors. Since no alternative plan can be found, the plan reports that the requested goal cannot be satisfied given the current circumstances.

Table V. Performance Results: Composition Times for Trivial Domains of Increasing Size

Actions in domain:		**34**	**68**	**136**	**204**
CSP construction time:		0.6 sec	0.6 sec	0.9 sec	1.1 sec
Goal a: turn on all lights	Plan size:	9	18	36	54
	Planing time:	0.7 sec	0.9 sec	1.1 sec	2.1 sec
Goal b: close all curtains	Plan size:	8	16	32	48
	Planning time:	0.8 sec	1 sec	1.5 sec	2.8 sec

6.2. Domains with Increasing Size

In order to give some impression about the scalability of the CSP-based planner with respect to the size of the service domain (i.e., the number of available service operations), as well as the size of the compositions (i.e., the number of actions in the plan) we have performed some tests with simple goals. The results are summarized in Table V. The planning domains consist of an increasing number of devices of type lights, curtains, and windows (the last two are interdependent as already described), each of which has two operations, turn on/off or open/close. The first goal specifies that all lights in the domain should be switched off, starting from a condition where all lights are on, and the second one that all curtains should be closed, starting from a state where all windows are open inwards. Thus, the number of actions required to satisfy the goal depends on the number of available device instances. The search strategy used in these tests selects the integer variable involved in the largest number of constraints.

It should be emphasized, however, that the performance of the planner is not that much affected by the size of the domain or the plan, but mainly depends on the structure of both the planning domain, that is, the interdependencies between the actions, and the goal. For example, disjunctive propositions resulting either from action preconditions or the goal (e.g., in cases where the under-condition goal construct is used), are known to add an extra burden to the constraint solver. In general, CSP-based planners [Barták and Toropila 2009] are not yet as competitive as the best-performing planners in International Planning Competitions, such as Lama [Richter and Westphal 2010], or SAT-based planners. On the other hand, CSP formalisms are very expressive, since constraints in the context of a multivalued encoding allow us to naturally go beyond logical formulas, and use arithmetic formulas in preconditions and effects without sacrificing efficiency.

It should be mentioned though, that realistic service domains are usually not as structurally complex as the grid domain or the domains used in the International Planning Competitions (for example, compare the broadly used travel domain with the PDDL domains in ipc.icaps-conference.org). In service environments, complexity does not usually stem from highly transitive interdependencies between actions/service operations, but challenges come from other sources, such as incomplete knowledge and sensing, the dynamic nature of context, output-to-input parameter passings regarding variables that range over large domains, and the support for extended goals. The high expressive power provided by the CSP-based planner comes a at the cost of computational efficiency. However, the main concern of the proposed planning framework is not achieving high performances in complex combinatorial reasoning, but rather demonstrating that it can successfully deal with a number of different scenarios in domotic application domains, through the provision of an intuitive representation that naturally fits the variable-based device-level world, and its capability to support extended goals, numeric variables, incorporation of dynamic context updates, and replanning for failure-handling.

In Table VI we present some results about the performance of the planner in a domain where a robot must move in a grid of adjacent rooms, interconnected through

Table VI.
Performance results: Planning time for a domain where a robot has to move between adjacent rooms by opening interconnecting rooms.

rooms/doors	**4 / 4**	**9 / 12**	**16 / 24**	**25 / 40**	**36 / 60**
Planning time (in sec) for goal i (move to target room)	1.1	1.9	3.6	12.8	65.8
Planning time (in sec) for goal ii (visit all rooms)	3.1	7.1	393	timeout	timeout

doors. The robot can open doors, and move to a neighboring room if the door leading to it is open. The first goal is a simple reachability goal which specifies a goal position that the robot should reach: the robot has to move from the uppermost left room of the grid to the bottom right one, starting from a condition where all doors are closed. The second goal dictates that the robot should visit at least once each room ($\wedge_{room_i \in Grid} achieve(robotLocRoom = room_i)$). It should be noted that the *achieve* goal construct corresponds to a large disjunction, which states that the proposition should be true in some of the planning states. The goals are issued in grid instances with an increasing number of rooms, and the planner aborts searching if no solution can be found within half an hour. The maximum planning length k is set to 2 times the number of rooms (note that due to the high degree of parallelism that characterizes the produced plans, many solutions which require considerably more than k actions will be found). The search strategy is based on selecting the variable involved in the largest number of static constraints and randomly assigning values to it, and restarts after an upper limit of backtracks is reached. We see that the "visit all rooms" goal cannot be satisfied for large domains consisting of more than 16 rooms.

7. USER EVALUATION

According to the ISO 9241-11 standard, usability refers to "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." In the case of the SM4ALL framework, the context of use is determined by the diverse requirements, abilities, and technological knowledge of the intended users. To perform a fair test, we identify two antithetic groups: the first group comprises elderly people, some of whom suffer severe motor disability, and will be referred to in the following as the Elderly and Disabled (E/D) group; the second group consists of young people experienced with computer innovations, who will from now on be referred to as the Technological Savvies (TS). The focus of the testing methodology is to assess whether the architectural design and implementation of the SM4ALL framework is useful and usable by users with diverse capabilities and aspirations, without the need of personalized reconfigurations.

The user evaluation methodology bases on a quantitative analysis regarding a number of basic dimensions, which are determined by connecting the established usability components in the literature [Nielsen 1994b; 1994a; Kim et al. 2003] with the context of domotic environments. Each of the dimensions takes into account some specific metric parameters, about which users are asked to give a score (usually in a scale from 0 to 4). The main dimensions' scores are calculated by taking the average of the parameters' values. In the following we list the main usability features along with their relevant metric components.

—*Acceptability* of domotic solutions in general captures the opinion of users towards the importance of domotics technology, their eagerness to delegate tasks to a computer, and their extent of concerns towards privacy intrusion.
—*Learnability* assesses how easy it is for the user to get familiarized with the system. It refers to the amount of effort the users have to make in order to comprehend the

functionalities of the system, and to be able to control it in accordance with the tasks they want to accomplish.

—*Aggregate system effectiveness* measures how satisfied the users are with the system, by taking into account a number of aspects referring to different components. Virtual apartment effectiveness refers to the extent to which the design of the home and the optical effects at the visualization level give the feeling of a real home. Two metrics are used with respect to the control panel's usability, assessing how clear and attractive the Web interface is, and how convenient to use it is. One more metric is used to assess the satisfaction of users with respect to the support of complex goals. Finally, the extent of difficulties or irritation resulting from some unexpected behavior or missing feature, and from low performance is also taken into account.

—*Efficiency* is concerned with the speed at which the system performs certain tasks. Time efficiency is measured with respect to the user's assessment of the time required to complete atomic operations and complex goals.

7.1. Experimental Setup

Demographics. The E/D group consists of 31 elderly people (12 males and 19 females), between 47 and 91 years old, and an average age of 71 years. Eight persons out of this group suffer chronic neurological disorders and make use of the BCI (5 males, 2 females; mean age $= 64.85 \pm 5.81$ years). All users of this group are clients of the Frisian health care institution in the Netherlands "Thuiszorg Het Friese Land" (THFL). Moreover, 13 of the users in this group have experience with computers, and 9 of them make use of some kind of domotic devices at their house (e.g., automatic shutters, motorized armchair, lights, etc.). The testing took place in the months of October and November 2010. The TS group consists of 30 students who are doing their MSc in computer science at the University of Groningen, and attend a course on ubiquitous computing in the spring of 2011. Their age ranges between 21 and 30 years old, with an average age of 25 years. 10% of them are female and 90% male. Naturally, all members of this group are advanced users of computers, and three of them have used actual domotic devices.

Testing sessions. Figure 8 provides a high-level overview of the essential components of the experimental setup, and the basic sequence of events that take place between them. The user issues his commands via the Web interface or the BCI panel, depending on whether he suffers motor disabilities or not. The instructions are passed to the lower levels of the SM4ALL platform, and the results are reflected at the visualization level, projected on a separate screen, while the Web interface view is updated accordingly. The home instance visualized and controlled by the users is based on a virtual reconstruction of a real apartment built at the premises of the Fondazione Santa Lucia in Rome. The apartment consists of four rooms (two bedrooms, a kitchen, and a living room), equipped with 32 simulated devices (lights, doors, motorized bed, curtains, windows, TV, air-conditioner, etc.). The Web interface provides icons for controlling individually all available devices, organized per room view, and additional icons for modeling two complex goals, one for adjusting the living room conditions for watching TV, and one for preparing the bedroom for sleeping. The BCI offers control capabilities for only a subset of the devices and goals, since it supports up to 16 icons at a time. Users are asked to follow an instructive scenario, that is, a predetermined set of actions specified by the experimenters, which includes achieving certain conditions, for example, preparing the house for the night, by issuing individual commands, such as "turn off kitchen light", and the complex goals that are made available to them. The user can then freely interact with the system. At the end, the user is asked to fill in a questionnaire, where she is required to provide a score for each of the usability components already described. Along with the metric values, users are encouraged to provide a short explanation of

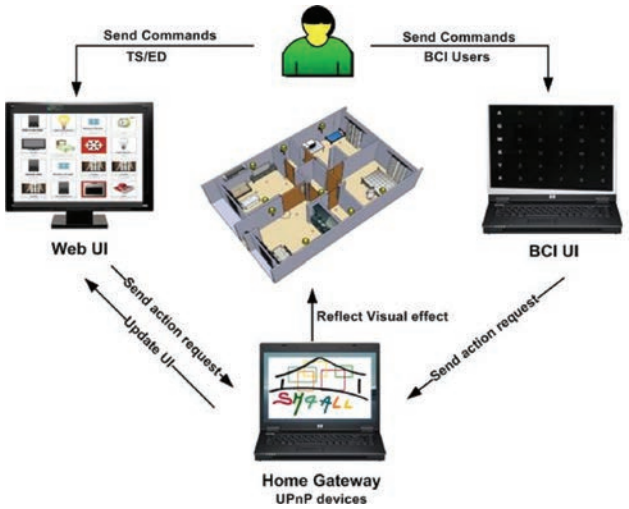

Fig. 8. Basic interactions between the components at the experimental setup.

their assessment. The questionnaire addressed to the TS group includes some extra questions with respect to the one addressed to the E/D group, requesting more details about the assessment of time efficiency and the effectiveness of complex goals.

The testing sessions with the E/D group have been arranged and conducted in cooperation with staff members of the THFL. The testing sessions which do not involve use of the BCI took place separately for each user, at his home of residence. Each individual testing session, including the platforms setup in the users' environment, lasted one hour in average. The BCI testing sessions took place at the THFL premises, conducted in two consecutive days. The first day was dedicated to the training of the BCI system, and making a profile of the brain activity of each participant (the BCI training requires 30 minutes on average per person). The second day the users were ready to interact with the actual SM4ALL platform. The tests with the TS group were conducted during three different sessions in a university lecture room.

7.2. Usability Evaluation Results

Elderly people. Table VII summarizes the quantitative findings of the usability tests with the members of the E/D group. All quantitative factors included in the questionnaires are mapped to a scale from 0 to 1. Results of time efficiency assessment or particular to the complex goals are not presented, as this was intended specifically for the TS group. A natural observation is that the amount of effort reported by the participants of the BCI tests is higher than the effort assessment of the users who did not have to learn how to use the BCI equipment. The findings indicate that satisfaction from the Web interface effectiveness is particularly high, while satisfaction from the virtual experience delivered by the visual effects is quite lower. The data are further analyzed in the comparative evaluation presented in at the end of this section.

Technological savvies. The quantitative results of the testing sessions conducted with the members of the TS group are presented in Table VIII. Similarly to the findings from the E/D group, the technological savvies gave a high score to the effectiveness of the control panel and a lower score to the satisfaction from the virtual home feeling. It is worth mentioning the particularly high assessment of the complex goals effectiveness, which the members of this group highlighted as particularly interesting and useful.

Table VII. Aggregated Results of the Usability Tests with the E/D Group

	Mean		
	Non-BCI	BCI	Overall
Acceptance of privacy disclosure [0 (negative) - 1 (positive)]	0.79	0.91	0.82
Acceptance of tasks automation	0.7	0.82	0.72
Aggregate domotics acceptance	0.78	0.88	0.8
Learnability effort [0(low) - 1 (high)]	0.15	0.37	0.26
Control panel effectiveness [0 (negative) - 1 (positive)]	0.88	0.93	0.89
Virtual home effectiveness	0.78	0.69	0.76
Aggregate framework effectiveness	0.83	0.79	0.82

Table VIII. Aggregated Results of the Usability Tests with the TS Group

	Mean
Acceptance of privacy disclosure [0 (negative) - 1 (positive)]	0.48
Acceptance of tasks automation	0.81
Aggregate domotics acceptance	0.74
Learnability effort [0(low) - 1 (high)]	0.4
Complex goals effectiveness [0 (negative) - 1 (positive)]	0.89
Virtual home effectiveness	0.7
Control panel effectiveness	0.84
Aggregate framework effectiveness	0.78
Time efficiency complex goals [0 (slow) - 1 (fast)]	0.79
Time efficiency for atomic actions	0.9
Aggregate time efficiency	0.83

Satisfaction from the time performance of tasks associated to complex goals is rather smaller with regard to efficiency of performing atomic operations. The diagram Figure 9 shows how satisfaction from time efficiency is distributed in the TS group. The results indicate that 76% of the users give a time efficiency rank of over 0.7, while only 3% of the users give an assessment lower than 0.5. Further analysis of the rest of the dimensions is provided in the next section.

Comparative analysis. Comparing the results of the two groups we can draw some interesting conclusions. Regarding the acceptability of domotic technologies, the E/D group is more reluctant to have a computer overtaking tasks, while the TS group is more positive towards the automation of domotic routines. Moreover, 30% of the E/D group gives a score below 0.5 to acceptance of domotic tasks automation. On the other hand, members of the E/D group express less concerns about privacy in comparison with the

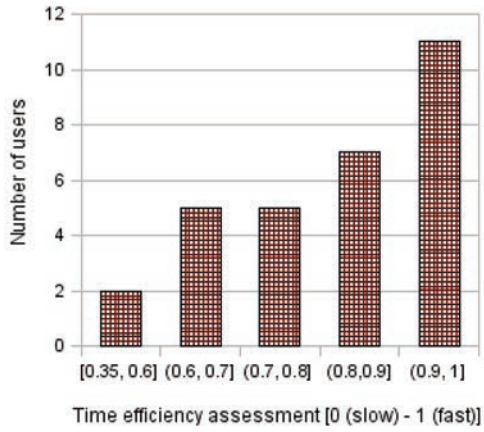

Fig. 9. Overall time efficiency assessment by the TS group.

TS group. Many of the elderly, and especially the ones suffering from serious disabilities or health problems, are quite used to being surveilled and looked after by specialized personnel, such as nurses or household assistants, and therefore 50% of them do not express any considerable worries about privacy intrusion. On the contrary, 40% of the young technological savvies are seriously concerned about invasion of privacy and violation of personal space.

With respect to the amount of time required for understanding and learning how to use the framework, members of the E/D group needed 10 min on average, while the technological savvies 2 min on average. For the members of the E/D group who had not used a computer before, considerable time was required to get familiarized with the use of a mouse. Although members of the E/D group presumably needed more time to learn the system, both groups assessed that the system was easy to perceive and control: 73% of the E/D and 83% of the TS users put the amount of the learnability effort between 0 and 0.25.

Regarding aggregated satisfaction from the system's effectiveness, it should be noted that in the case of the E/D group the average is in most cases calculated with respect to less constituting parameters, because the members of the E/D group left many questions unanswered. As can be seen in the distributions plotted in Figure 10, the findings regarding the TS group can be approximated by a Gaussian distribution, with a mean of $\mu = 0.78$ and $\sigma = 0.15$. In the case of the E/D group, the population is concentrated around higher values of aggregate effectiveness assessment, with 65% giving a value of over 0.8. In both groups, for 77% of the users, the aggregate assessment for effectiveness is over 0.7.

8. RELATED WORK

The main technical result of the present manuscript is a designed, implemented, and evaluated architecture based on the concept of dynamic service composition in a pervasive computing environment. Therefore we consider as related work service composition, especially with respect to pervasive systems, research on the Web of Things and finally we overview domotics project close to ours in spirit.

8.1. Service Composition in Pervasive Systems

A great number of approaches have been proposed in the literature about describing, constructing, and executing Web services compositions, with a lot of research performed

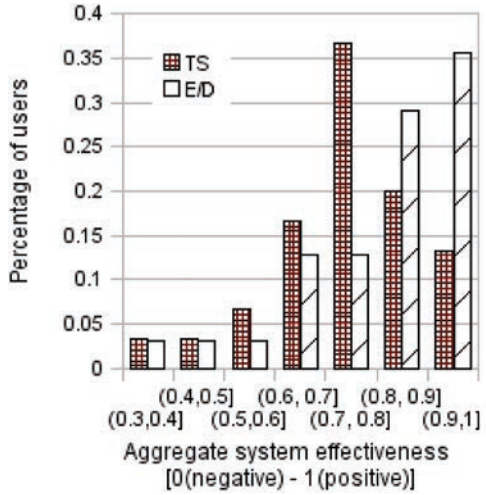

Fig. 10. Comparative distribution of aggregate assessment with respect to system's effectiveness.

from different viewpoints such as workflow management, the semantic Web, and model checking. A lot of associated challenges have to be tackled for efficient service compositions to be realized, related to service discovery and matchmaking, for example, Skoutas et al. [2008] and Pilioura and Tsalgatidou [2009], support for process evolution and migration, for example, Ryu et al. [2008] and Orriëns and Yang [2006], and QoS awareness, for example, Yu et al. [2007].

BPEL, the standard for expressing service compositions, has been proposed before to guide the coordination of pervasive systems. In Lazovik et al. [2009] we used a BPEL engine to demonstrate some complex scenarios on the visualization platform. In Redondo et al. [2008], composite services deployed as BPEL processes are made available in a semantically enriched OSGi platform. However, BPEL processes are precompiled and thus support limited dynamicity. In Etzioni et al. [2010] BPEL processes in a smart home are enhanced with a runtime fault management mechanism, where the receipt of a fault-indicating event triggers an appropriate predefined fault template according to the semantically inferred type of the fault. However, these approaches do not overcome the limited flexibility and adaptability deriving from the rigid nature of predefined processes.

To move towards more flexible and context-aware coordination mechanisms, AI planning methodologies have been proposed for automatically composing semantic services in the Web, for example, Agarwal et al. [2005], Au et al. [2005], and Sohrabi et al. [2006]. Most of the approaches to service composition via automated planning, however, require that the set of supported solutions is predefined in some form of procedural templates, such as in the form of HTN methods in Au et al. [2005] or as Golog programs in Sohrabi et al. [2006], and are therefore not easily reconfigurable in case of changes in the context, the domain, or the user requirements. Our approach, on the other hand, relies on a domain-independent planner (see Kaldeli et al. [2011] for details) where the user just states what properties have to be satisfied, without having to anticipate how these can be fulfilled. The compositions computed in Agarwal et al. [2005] are also computed by a domain-independent planner, which, however, requires a grounded representation of the planning domain, and thus comes short in the efficient handling of variables ranging over large domains.

In the context of domotic systems, AI-inspired techniques have been used for coordinating intelligent components in a ubiquitous environment, without, however, emphasizing on services in any specific way. In Pecora and Cesta [2007], each device is represented as a software agent and the problem of service integration is cast to distributed constraint optimization. The coordination takes place in a purely distributed manner, relying on the communication between independent agents. On the negative side, modeling the home behavior involves specifying all possible inter-relations between the variables comprising the domain in terms of complex constraints, which makes it a fairly cumbersome process, even for a limited number of services. The requests the user can make to the system are limited to a set of rather simple commands, which only involve the interaction of a limited set of predefined agents. A hierarchical task network planning approach is adopted in Gravot et al. [2006] for controlling a humanoid robot so that it performs certain cooking tasks. The planner bases on the description of predefined methods expressed in terms of the actions the robot can perform. A multiagent approach is adopted by Davidsson and Boman [2005] to control a smart building's conditions with the objective of saving energy and increasing users' satisfaction. This approach, however, focuses on triggering some predefined rules as a reaction to certain events rather than on computing complex compositions of different services. In Aker et al. [2011] the action description language C+ is used for modeling a housekeeping domain: multiple robots have to collaborate to tidy up by moving objects around the house, and the causal reasoner CCALC is applied to plan the robots' activities in a safe way.

8.2. The Web of Things

The Web of Things is a term to describe Web-like infrastructures where the interconnected objects can be physical ones, for which there is a virtual representation in the software architecture. These objects can be physically accessed and manipulated by human beings. Wireless sensors, embedded devices, or RFID-tagged items are integrated into a pervasive network and can communicate with other objects and services using Web-based principles, from SOAP and WSDL to Ajax and REST. The Cool Town project [Kindberg et al. 2000] is one of the first examples proposing the application of the Web paradigm for interlinking physical objects. These interact by exchanging messages via HTTP connections and by the use of a standard interface, rather than having heavy middleware applications running on each device. The standard Web technologies are extended to support discovery, mobility, and location awareness, and devices are indexed via Web pages which make their services available to users.

The principle of RESTful services is broadly used for providing a uniform HTTP interface to interacting with smart things, independent of their platform protocol. Duquennoy et al. [2009] demonstrate that putting Web servers directly on resource-constrained devices is a feasible solution. The authors of Trifa et al. [2010], on the other hand, argue for the use of smart gateways, which hide the underlying specific network protocols of the connected devices, and can thus be used for providing aggregate functions, based, for example, on composing single lower-level services. An extended discussion of different approaches building upon Web principles is provided in Guinard et al. [2011], where it is shown how the notion of Web mashups can be applied to physical objects, in order to offer more customization possibilities to end-users. Since the focus of the present treatment is on realizing home smartness via dynamic service composition, independently of the underlying invocation mechanism, we do not enter into the debate of RESTful versus Web-service-based architectures.

In recent years, several SOAs, such as UPnP and Jini, have emerged to provide interoperability with minimum human intervention. The OSGi platform has been widely

used as a platform- and application-independent residential gateway that enables interconnection, discovery, and coordination of different devices, thus offering more flexibility to domain designers, for example, Zeadally and Kubher [2008] and LEE et al. [2009]. Moving towards a semantic annotation of the OSGi description is proposed in Gouvas et al. [2007] to improve the discovery process. Aiello [2006] investigate the use of Web services in the domestic network, and in Aiello and Dustdar [2008] the application of the Web service stack is proposed as a means to solve the interoperability problem at home. The architecture builds on using WS-notification as an event-based mechanism for addressing emergency situations in the home, most notably the fall of an elder, however, the aspects of context and coordination of service are not addressed beyond the basic action/reaction interactions. In Cabezas et al. [2008], an architecture for extending the OSGi registry with semantic terms is proposed, which allows the automatic parsing of services by software agents. However, no tasks more complex than service registration and invocation are considered.

8.3. Smart Homes Projects

A number of research and industrial projects focusing on supporting a wide range of household devices over heterogeneous network environments have been performed and are underway. In the following we present a brief account of such projects that are most close in spirit to the SM4ALL aspects of focus. SOCRADES [Spiess et al. 2009] focuses on an SOA-based integration architecture which enables the collaboration of ubiquitous devices in the manufacturing domain with services offered at the enterprise application level. Like in our case, Web services are embedded to devices and a publish-subscribe mechanism is used to handle events. However, only execution of predefined descriptions of service compositions, such as BPEL or mashups, is supported, and runtime flexibility is limited to selecting the right instances for a fixed sequence of service types.

HYDRA [Eisenhauer et al. 2010; Zhang and Hansen 2008] proposes a service-oriented middleware platform for networked embedded systems, which supports a model-driven development of ambient intelligence applications, based on ontologies of semantic devices. Semantic rules are used for diagnosing possible malfunctioning in the system, however, there is no support for intelligent composition generation to deal with such situations. The RUNES middleware [Costa et al. 2007] for embedded systems requires explicit connectors between components which may have inter-related functionalities, and which can be further organized into groups that can form stacks of overlay services. This design leads to a layered architecture, however, with limited dynamicity and no automatic reasoning capabilities. The SM4ALL platform tackles the problem of home automation at a higher application level, and focuses on dynamic and runtime compositions of embedded services connected to the network, thus realizing a system that is more user centric, customizable, and context-adaptable with respect to the aforementioned infrastructures.

Several approaches stimulated from the field of artificial intelligence have been adopted by projects that seek to leverage the smartness exposed by homes equipped with smart sensors and actuators. In the context of the MavHome project, learning algorithms are employed in Rao and Cook [2004] to predict the occurrence of common activities that take place in a home, and decide whether it should take them over automatically. The intelligent buildings project, for example, Davidsson and Boman [2005], builds upon an agent-based approach which has already been discussed in Section 8.1. In the course of the ThinkHome project, the use of neural networks is proposed in Kastner et al. [2010] to learn the optimal values for the parameters of automation activities with respect to context and user preferences, such as specifying the best time to start heating a room based on weather conditions.

9. CONCLUSIONS

Paradigms and techniques that have bloomed in the area of the Internet Web are general and applicable to other domains. Environments with loosely coupled computational hosts, running heterogeneous hardware and software, associating one another in a just-in-time manner are characteristic of several domains. Pervasive computational environments, such as our future homes, are a prominent example of this trend. In this article, we consider the highly dynamic case of a smart home, where devices join and leave spontaneously, and the user interacts with them by expressing high-level goals, rather than sending individual invocations to the devices. The approach is based on a service-oriented architecture to address heterogeneity and late binding, and takes advantage of state-of-the-art AI planning techniques to address the issues of service composition and context awareness. The overall result is a generic and customizable middleware for the home that is able to deal with the dynamic nature of a physical environment inhabited by humans.

The approach is fully implemented in a proof-of-concept system, which has been tested and evaluated, in its simulated version, both from the technical point of view as well as by is potential users. A deployment in a physical home with actual devices is currently underway to achieve even better evaluation of the users' appreciation of such a system. The results of the technical evaluation show that the generic service-oriented architectural design, OSGi/UPnP device wrapping, asynchronous notification exchange, and automated planning for context-aware composition is a feasible and realistic approach to pervasive computing at home. The usability tests indicate that the assessment of the SM4ALL prototype is positive, with users being highly satisfied by the design and usability of the UI, as well as the support of complex goals, while expressing some reservations about the extent of the virtual home experience. Both elderly and disabled users as well as technological savvies find the platform easy and convenient to understand and control. The test findings also indicate that despite the use of time- and resource-consuming techniques such as advanced AI planning or the visualization tool, from the users' point of view the system is quite efficient in terms of the time required to accomplish simple and more complex tasks. We report on one particular user that due to a neural decease can only communicate with a joystick, and who was particularly enthusiastic about the potential of the SM4ALL middleware. Being a journalist, he has provided a written account of his experience of using the BCI for expressing goals to control a virtual home [Haisma 2011].

From the technical point of view, the approach to service composition is more general than the specific smart home application shown here. Any situation necessitating to automatically and on-the-fly combine resources that are formally described is amenable to the approach presented. Interesting examples are supply chains, for example, Lazovik et al. [2003] service-based virtual enterprises, such as, Mehandijev and Grefen [2011], and services on the public Web [Kaldeli et al. 2011; Hassine et al. 2006; Skoutas et al. 2008]. We claim, though, that smart homes constitute an environment that is particularly expedient for the application of AI planning techniques. In fact, the applicability of elaborate automated discovery and composition of services available online in the Web is limited by the lack of machine-interpretable and standardized semantic markups, as well as the very limited meaningful correlation and compatibility among operations of different services, with the vast majority of public services being mere data sources, as concluded by the findings presented in Fan and Kambhampati [2005]. The environment of a smart home, on the other hand, is more structured, well-defined, and controllable, thus making the added value gained by nontrivial automated composition and monitoring of services a feasible and realistic task. In this case, one can rely on consistent descriptions of service operations, with proper syntactic and semantic

markups provided by the home domain designer, to perform powerful reasoning for complex tasks which considerably advance the level of home intelligence.

The present work, beside pointing to directions for further research developments in the areas of pervasive systems, human-computer interaction, security, and sociology, also brings forth interesting research questions in the general area of Web systems. From this perspective, the general dealing of concurrency is crucial in the massively parallel environment of the home, where more users are "executing" while the system is running. Dealing with concurrency and possible contradictions that may arise when events interfere with the execution of a plan is essential to move to a product. For example, considering a plan that prescribes first opening a door and then moving the robot through it, some user or some other orchestration running in parallel may close the door just after the orchestrator has invoked its opening, and before the robot has moved through it. Moreover, some services may be characterized by a byzantine behavior, and result in arbitrary effects different than those prescribed in their semantic description. In order to address such situations, the validity of the plan has to be reconfirmed at each step of the execution, so as to identify potential inconsistencies and replan to recover from undesirable outcomes if necessary. The development of an effective and efficient continual planning algorithm that can handle such unforeseen contingencies is a top priority in our future work.

Another issue that has to be addressed is the undoing of operations. Rolling-back the effects of actions becomes relevant under several situations. For example, the user may change his mind after issuing a goal, and while a composition is in the middle of execution request its canceling. In such a case, the effects of all actions that were executed as part of the goal have to be undone. Moreover, when replanning to recover from some failure, undesirable situations can be the result of certain actions which were performed as part of the old plan. If not undone, such actions may lead to unnecessary energy consumption or even turn out to be dangerous (e.g., the gas may be left on). Backtracking and compensation mechanisms while keeping the need for manual intervention to the minimum is a demanding challenge which goes beyond the scope of this article. Policies similar to the compensation handlers used in BPEL processes [Ma et al. 2009] may be relevant. A straight-forward way to partly deal with this issue for the scenarios considered herein is to keep a predefined list of actions whose effects are estimated to be critical (e.g., turning on the oven), together with their respective "undo" action, which can roll back their effect (e.g., "turnOnOven" and "turnOffOven").

From the users point of view, we consider that those with a basic knowledge of computing technologies are empowered with the capability to customize the domotic platform themselves, by articulating their requirements in terms of the expressive and declarative goal language. To facilitate this for them, the assistance of a goal editor, which allows the user to combine the set of desired properties by means of the available high-level goal constructs, is indispensable and we are currently investigating solutions for this. Regarding the process of transforming the pervasive-level services to planning-level actions, developing an ontology of devices that provides the necessary semantic descriptions in a well-defined hierarchy would greatly facilitate the work of the domain designer. Our precondition and effects language is in spirit with existing semantic markups for Web services such as OWL-S, which can be inserted into the OSGi registry similarly as in Cabezas et al. [2008]. Therefore, finding a suitable and yet powerful interface for designing goals and service descriptions is open for future investigation. In summary, smart homes are ever-more in sight and can be seen as an internet of interoperating devices. Thus, experience and solutions from the Internet Web will be most valuable for these systems and perhaps will see the merging of the two types of Internet in the long run.

ACKNOWLEDGMENTS

The authors would like to thank the partners of the SM4All project and THFL, in particular for their support in the user evaluation sessions. We are also thankful to our students Jaap Bresser, Elena Lazovik, Allard Naber and Sardar Yumatov for their software contributions.

REFERENCES

AGARWAL, V., DASGUPTA, K., KARNIK, N., KUMAR, A., KUNDU, A., MITTAL, S., AND SRIVASTAVA, B. 2005. A service creation environment based on end to end composition of web services. In *Proceedings of the 14th International Conference on World Wide Web (WWW)*. ACM Press, New York, 128–137.

AIELLO, M. 2006. The role of web services at home. In *Proceedings of the IEEE Conference on Web Service-Based Systems and Applications (WEBSA)*. IEEE Computer Society, 164.

AIELLO, M. AND DUSTDAR, S. 2008. A domotic infrastructure based on the web service stack. *Pervasive Mobile Comput. 4*, 4, 506–525.

AIELLO, M., PAPAZOGLOU, M., YANG, J., CARMAN, M., PISTORE, M., SERAFINI, L., AND TRAVERSO, P. 2002. A request language for web-services based on planning and constraint satisfaction. In *Proceedings of the 3rd International Workshop on Technologies for E-Services (TES)*. Lecture Notes in Computer Science, vol. 2444, Springer, 76–85.

AKER, E., ERDOGAN, A., ERDEM, E., AND PATOGLU, V. 2011. Causal reasoning for planning and coordination of multiple housekeeping robots. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*. 311–316.

ALOISE, F., SCHETTINI, F., ARICO, P., BIANCHI, L., RICCIO, A., MECELLA, M., BABILONI, F., MATTIA, D., AND CINCOTTI, F. 2010. Advanced brain computer interface for communication and control. In *Proceedings of the International Conference on Advanced Visual Interfaces*. ACM Press, New York, 399–400.

ALOISE, F., SCHETTINI, F., ARICO, P., SALINARI, S., GUGER, C., RINSMA, J., AIELLO, M., MATTIA, D., AND CINCOTTI, F. 2011. Asynchronous p300-based bci to control a virtual environment: Initial tests on end users. *Clinical EEG Neurosci. 42*, 4, 1–6.

AU, T., KUTER, U., AND NAU, D. 2005. Web service composition with volatile information. In *Proceedings of the 4th International Semantic Web Conference (ISWC)*. 52–66.

BARTA K, R. AND TOROPILA, D. 2009. Enhancing constraint models for planning problems. In *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference*. AAAI Press.

BERARDI, D., CALVANESE, D., GIACOMO, G. D., LENZERINI, M., AND MECELLA, M. 2005. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst. 14*, 4, 333–376.

BRONSTED, J., HANSEN, K. M., AND INGSTRUP, M. 2010. Service composition issues in pervasive computing. *IEEE Pervasive Comput. 9*, 62–70.

CABEZAS, P., ARRIZABALAGA, S., SALTERAIN, A., AND LEGARDA, J. 2008. An agent-based semantic osgi service architecture. In *Computer and Information Science. Studies in Computational Intelligence Series*, vol. 131, Springer, 97–106.

CARUSO, M., CICCIO, C. D., IACOMUSSI, E., KALDELI, E., LAZOVIK, A., AND MECELLA, M. 2012. Service ecologies for home/building automation. In *Proceedings of the 10th International IFAC Symposium on Robot Control*.

CATARCI, T., DI CICCIO, C., FORTE, V., IACOMUSSI, E., MECELLA, M., SANTUCCI, G., AND TINO, G. 2011. Service composition and advanced user interfaces in the home of tomorrow: The sm4all approach. In *Proceedings of the 2nd International ICST Conference on Ambient Media and Systems (Ambi-Sys'11)*.

COSTA, P., COULSON, G., MASCOLO, C., MOTTOLA, L., PICCO, G., AND ZACHARIADIS, S. 2007. Reconfigurable component-based middleware for networked embedded systems. *Int. J. Wireless Inf. Netw. 14*, 149–162.

DAVIDSSON, P. AND BOMAN, M. 2005. Distributed monitoring and control of office buildings by embedded agents. *Inf. Sci. 171*, 293–307.

DUQUENNOY, S., GRIMAUD, G., AND VANDEWALLE, J.-J. 2009. The web of things: Interconnecting devices with high usability and performance. In *Proceedings of the 6th International Conference on Embedded Software and Systems (ICESS)*. IEEE Computer Society, 323–330.

EISENHAUER, M., ROSENGREN, P., AND ANTOLIN, P. 2010. HYDRA: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *Proceedings of the Workshop on the Internet of Things: 20thTyrrhenian Workshop on Digital Communications*. Springer, 367–373.

ETZIONI, Z., KEENEY, J., BRENNAN, R., AND LEWIS, D. 2010. Supporting composite smart home services with semantic fault management. In *Proceedings of the 5th International Conference on Future Information Technology (FutureTech)*. 1–8.

FAN, J. AND KAMBHAMPATI, S. 2005. A snapshot of public web services. *SIGMOD Rec. 34*, 1, 24–32.

GHALLAB, M., NAU, D., AND TRAVERSO, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Fransisco, CA.

GOUVAS, P., BOURAS, T., AND MENTZAS, G. 2007. An osgi-based semantic service-oriented device architecture. In *Proceedings of the OTM Workshop on On the Move to Meaningful Internet Systems*. 773–782.

GRAVOT, F., HANEDA, A., OKADA, K., AND INABA, M. 2006. Cooking for humanoid robot, a task that needs symbolic and geometric reasonings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 462–467.

GUGER, C., DABAN, S., SELLERS, E. HOLZNER, C., KRAUSZ, G. CARABALONA, R., GRAMATICA, F., AND EDLINGER, G. 2009. How many people are able to control a p300-based brain-computer interface (bci)? *Neurosci. Lett. 462*, 94–98.

GUINARD, D., TRIFA, V., MATTERN, F., AND WILDE, E. 2011. From the internet of things to the web of things: Resource oriented architecture and best practices. In *Architecting the Internet of Things*. Springer, 97–129.

HAISMA, H. 2011. From thoughts to actions. http://www.cs.rug.nl/_aiellom/publications/hasima.pdf.

HASSINE, A. B., MATSUBARA, S., AND ISHIDA, T. 2006. A constraint-based approach to horizontal web. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*. 130–143.

HELMERT, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artif. Intell. 173*, 503–535.

KALDELI, E. 2009a. Using csp for adaptable web service composition. Tech. rep. 2009-7-01, University of Groningen. www.cs.rug.nl/_eirini/tech rep 09-7-01.pdf.

KALDELI, E. 2009b. Using CSP for adaptable web service composition. Tech. rep. 2009-7-01, Johann Bernoulli Institute, University of Groningen.

KALDELI, E., LAZOVIK, A., AND AIELLO, M. 2009. Extended goals for composing services. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

KALDELI, E., LAZOVIK, A., AND AIELLO, M. 2011. Continual planning with sensing for web service composition. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. AAAI Press.

KALDELI, E., WARRIACH, E. U., BRESSER, J., LAZOVIK, A., AND AIELLO, M. 2010. Interoperation, composition and simulation of services at home. In *Proceedings of the 8th International Conference on Service Oriented Computing (ICSOC)*. Lecture Notes in Computer Science, vol. 6470, Springer, 167–181.

KASTNER, W., KOFLER, M. J., AND REINISCH, C. 2010. Using ai to realize energy efficient yet comfortable smart homes. In *Proceedings of 8th IEEE International Workshop on Factory Communication Systems (WFCS'10)*. 169–172.

KIM, S. H., KIM, S. W., AND PARK, H. 2003. Usability challenges in ubicomp environment. In *Proceedings of the International Ergonomics Association (IEA)*.

KINDBERG, T., BARTON, J., MORGAN, J., BECKER, G., CASWELL, D., DEBATY, P., GOPAL, G., FRID, M., KRISHNAN, V., MORRIS, H., SCHETTINO, J., AND SERRA, B. 2000. People, places, things: Web presence for the real world. In *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMSCA)*. 365–376.

LAZOVIK, A., AIELLO, M., AND PAPAZOGLOU, M. 2003. Planning and monitoring the execution of web service requests. In *Proceedings of the 1st International Conference on Service-Oriented Computing (ICSOC'03)*. Lecture Notes in Computer Science, vol. 2910, Springer, 335–350.

LAZOVIK, E., DEN DULK, P., DE GROOTE, M., LAZOVIK, A., AND AIELLO, M. 2009. Services inside the smart home: A simulation and visualization tool. In *Proceedings of the 7th International Conference on Service-Oriented Computing (ICSOC-ServiceWave'09)*. Lecture Notes in Computer Science, vol. 5900, Springer, 651–652.

LEE, C., KO, S., KIM, E., AND LEE, W. 2009. Enriching osgi service composition with web services. *IEICE Trans. Inf. Syst. E92.D*, 5, 1177–1180.

LI, Q., STANKOVIC, J. A., HANSON, M. A., BARTH, A. T., LACH, J., AND ZHOU, G. 2009. Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information. In *Proceedings of the 6th International Workshop on Wearable and Implantable Body Sensor Networks*. 138–143.

MA, C., XU, Q., AND SANDERS, J. W. 2009. A survey of business process execution language (BPEL). Tech. rep. 2009-7-01, UNU-IIST.

MEHANDIJEV, N. AND GREFEN, P., EDS. 2011. *Dynamic Business Process Formation for Instant Virtual Enterprises*. Springer.

NIELSEN, J. 1994a. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence*. ACM Press, New York, 152–158.

NIELSEN, J. 1994b. *Heuristic Evaluation. Usability Inspection Methods*. John Wiley and Sons.

ORRIENS, B. AND YANG, J. 2006. A rule driven approach for developing adaptive service oriented business collaboration. In *Proceedings of the IEEE International Conference on Services Computing*. 182–189.

Pecora, F. and Cesta, A. 2007. DCOP for smart homes: A case study. *Comput. Intell. 23*, 4, 395–419.

Pilioura, T. and Tsalgatidou, A. 2009. Unified publication and discovery of semantic web services. *ACM Trans. Web 3*, 3, 11:1–11:44.

Rao, S. P. and Cook, D. J. 2004. Predicting inhabitant action using action and task models with application to smart homes. *Int. J. Artif. Intell. Tools 13*, 81–100.

Redondo, R. P. D., Fernandez, V. A., Cabrer, M. R., Arias, J. J. P., Duque, J. G., and Solla, A. G. 2008. Enhancing residential gateways: A semantic osgi platform. *IEEE Intell. Syst. 23*, 1, 32–40.

Richter, S. and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. 39*, 127–177.

Ryu, S. H., Casati, F., Skogsrud, H., Benatallah, B., and Saint-Paul, R. 2008. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Trans. Web 2*, 13:1–13:46.

Skoutas, D., Sacharidis, D., Simitsis, A., and Sellis, T. 2008. Serving the sky: Discovering and selecting semantic web services through dynamic skyline queries. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing*. 222–229.

Sohrabi, S., Prokoshyna, N., and Mcilraith, S. A. 2006. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC)*. 597–611.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., De Souza, L. M. S., and Trifa, V. 2009. Soa-based integration of the internet of things in enterprise services. In *Proceedings of the 7th IEEE International Conference on Web Services (ICWS)*. 968–975.

Trifa, V., Guinard, D., Davidovski, V., Kamilaris, A., and Delchev, I. 2010. Web messaging for open and scalable distributed sensing applications. In *Proceedings of the 10th International Conference on Web Engineering (ICWE)*. Springer, 129–143.

Warriach, E. U., Kaldeli, E., Bresser, J., Lazovik, A., and Aiello, M. 2010. A tool for integrating pervasive services and simulating their composition. In *Proceedings of the 8th International Conference in Service-Oriented Computing (ICSOC).* Lecture Notes in Computer Science, vol. 6470, Springer, 726–727.

Yu, T., Zhang, Y., and Lin, K.-J. 2007. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web 1*, 1.

Zeadally, S. and Kubher, P. 2008. Internet access to heterogeneous home area network devices with an osgi-based residential gateway. *Int. J. Ad Hoc Ubiquitous Comput. 3*, 48–56.

Zhang, W. and Hansen, K. M. 2008. Semantic web based self-management for a pervasive service middleware. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. 245–254.

Received October 2011; revised October 2012; accepted December 2012